IJSC Editorial Board

Editors-in-Chief
Andrzej Goscinski, Deakin University, Australia
Jia Zhang, Carnegie Mellon University – Silicon Valley, USA

Associate Editor-in-Chief
Zibin Zheng, The Chinese University of Hong Kong

Editorial Board
Boualem Benatallah, University of New South Wales, Australia
Rajdeep Bhowmik, Cisco Systems Inc., USA
Irena Bojanova, University of Maryland University College, USA
Hong Cai, ZTE, USA
Rong Chang, IBM T.J. Watson Research Center, USA
Shiping Chen, CSIRO, Australia
Chi-Hung Chi, CSIRO, Australia
Ying Chen, Wisedu Information Technology, China
Ernesto Damiani, University of Milan, Italy
Schahram Dustdar, Vienna University of Technology, Austria
Ephraim Feig, Independent Consultant, USA
Aditya Ghose, University of Wollongong, Australia
Claude Godart, Univ. Nancy, Lorraine Univ., France
Nils Gruschka, University of Applied Science Kiel, Germany
Mohand-Said Hacid, Université Claude Bernard Lyon 1, France
Jun Han, Swinburne University of Technology, Australia
Akhil Kumar, Pennsylvania State University, USA
Patrick Hung, University of Ontario, Canada
San-Yih Hwang, National Sun Yat-sen University, Taiwan
Geetika Lakshmanan, IBM T.J. Watson Research Center, USA
Donghui Lin, Kyoto University, Japan
Shiyong Lu, Wayne State University, USA
Michael Lyu, The Chinese University of Hong Kong, Hong Kong
Stephan Marganiec, University of Leicester, UK
Louise Moser, University of California Santa Barbara, USA
Lorenzo Mossucca, Istituto Superiore Mario Boella(ISMB), Italy
Sushil Prasad, Georgia State University, USA
Lakshmish Ramaswamy, University Of Georgia, USA
Krishna Ratakonda, IBM Watson Research Center, USA
Anna Ruokonen, Penn State University, USA
George Spanoudakis, City University London, UK
Vijay Varadharajan, Macquarie University, Australia
Wei Tan, IBM T.J. Watson Research Center, USA
Mingdong Tang, Hunan University of Science and Technology, China
Hongbing Wang, Southeast University, China
Shanguang Wang, Beijing University of Posts and Telecommunications, China
Yan Wang, Macquarie University, Australia
Jian Wu, Zhejiang University, China
Xiaofei Xu, Harbin Institute of Technology, China
Yuhong Yan, Concordia University, Canada
Bing Bing Zhou, The University of Sydney, Australia

http://www.hipore.com/ijsc
### Table of Contents

#### iii. EDITOR-IN-CHIEF PREFACE
Andrzej Goscinski, Deakin University, Australia  
Jia Zhang, Carnegie Mellon University – Silicon Valley, USA

#### vi. Call for Articles: IJSC Special Issue of Application-Driven Services Innovations

#### RESEARCH ARTICLES

1. **Using Syntactic and Semantic Similarity of Web APIs to Estimate Porting Effort**  
   Hiranya Jayathilaka, UC Santa Barbara, USA; Alexander Pucher, UC Santa Barbara, USA  
   Chandra Krintz, UC Santa Barbara, USA; Rich Wolski, UC Santa Barbara, USA

15. **Cost-Effective Service Network Planning for Mass Customization of Services**  
   Zhongjie Wang, Harbin Institute of Technology, China; Nan Jing, Harbin Institute of Technology, China  
   Fei Xu, Harbin Institute of Technology, China; Xiaofei Xu, Harbin Institute of Technology, China

32. **An Analysis and Comparison of Cloud Data Center Energy-Efficient Resource Management Technology**  
   Zhihui Lu, Fudan University, China; Soichi Takashige, Fudan University, China  
   Soichi Takashige, Hitachi, Ltd., Japan  
   Yumiko Sugita, Fudan University, China; Tomohiro Morimura, Fudan University, China  
   Tomohiro Morimura, Hitachi, Ltd., Japan  
   Yutaka Kudo, Fudan University, China

52. **Modeling and Analysis of Mobile Push Notification Services Using Petri Nets**  
   Junhua Ding, East Carolina University, USA  
   Wei Song, Nanjing University of Sci. & Tech., China  
   Dongmei Zhang, China University of Geosciences, China

65. **Reference Architectures for Privacy Preservation in Cloud-based IoT Applications**  
   Ivor D. Addo, Marquette University, USA  
   Sheikh I. Ahamed, Marquette University, USA  
   Stephen S. Yau, Arizona State University, USA  
   Arun Buduru, Arizona State University, USA

http://www.hipore.com/ijsc
Editor-in-Chief Preface:

Services Discovery and Management

Andrzej Goscinski, Deakin University, Australia
Jia Zhang, Carnegie Mellon University – Silicon Valley, USA

Welcome to International Journal of Services Computing (IJSC). From the technology foundation perspective, Services Computing covers the science and technology needed for bridging the gap between Business Services and IT Services, theory and development and deployment. All topics regarding Web-based services lifecycle study and management align with the theme of IJSC. Specially, we focus on:
1. Web-based services featuring Web services modeling, development, publishing, discovery, composition, testing, adaptation, and delivery; and Web services technologies as well as standards;
2. Services innovation lifecycle that includes enterprise modeling, business consulting, solution creation, services orchestration, optimization, management, and marketing; and business process integration and management;
3. Cloud services featuring modeling, developing, publishing, monitoring, managing, and delivering XaaS (everything as a service) in the context of various types of cloud environments; and
4. Mobile services featuring development, publication, discovery, orchestration, invocation, testing, delivery; and certification of mobile applications and services.

IJSC is designed to be an important platform for disseminating high quality research on above topics in a timely manner and provide an ongoing platform for continuous discussion on research published in this journal. To ensure quality, IJSC only considers expanded version of papers presented at high quality conferences, key survey articles that summarizes the research done so far and identifies important research issues, and some visionary articles. At least two IJSC Editorial Board members will review the extended versions. Once again, we will make every effort to publish articles in a timely manner.

This issue collects the extended versions of five papers published at IEEE International Conference on Web Services (ICWS) and IEEE International Conference on Services Computing (SCC) in the general area of services discovery and management.

The first article is titled “Using Syntactic and Semantic Similarity of Web APIs to Estimate Porting Effort” by Jayathilaka and Krintz. The authors study an automated methodology for analyzing API similarity and quantifying the porting effort associated with the use of web APIs.

The second article is titled, “Cost-Effective Service Network Planning for Mass Customization of Services” by Wang and Xu. The authors use Service Network (SN) to represent a customizable composite service, so that MC of services is transformed into a Service Network Planning (SNP) problem.

The third article is titled, “An Analysis and Comparison of cloud Data Center Energy-Efficient Resource Management Technology” by Lu, Takashige, Sugita, Morimura and Kudo. The authors provide a survey of current industry and academic efforts on cloud data center energy-efficient management technology, focusing on the cloud data center resource dynamic provisioning technology and resource consolidation technology.

The fourth article is titled “Modeling and Analysis of Mobile Push Notification Services Using Petri Nets” by Ding, Song and Zhang. The authors proposed an approach to model mobile
computing services using a high level Petri nets and analyze them through combining formal verification and testing techniques.

The fifth article is titled “Reference Architectures for Privacy Preservation in Cloud-based IoT Applications” by Addo, Ahamed, Yau and Buduru. The authors present reference software architectures for building cloud-enabled IoT applications in support of collaborative pervasive systems aimed at achieving trustworthiness among end-users in IoT scenarios.

We would like to thank the authors for their efforts in delivering these five quality articles. We would also like to thank the reviewers, as well as the Program Committee of IEEE ICWS and SCC for their help with the review process.

About the Publication Lead

Liang-Jie (L.J) Zhang is Senior Vice President, Chief Scientist, & Director of Research at Kingdee International Software Group Company Limited, and director of The Open Group. Prior to joining Kingdee, he was a Research Staff Member and Program Manager of Application Architectures and Realization at IBM Thomas J. Watson Research Center as well as the Chief Architect of Industrial Standards at IBM Software Group. Dr. Zhang has published more than 140 technical papers in journals, book chapters, and conference proceedings. He has 40 granted patents and more than 20 pending patent applications. Dr. Zhang received his Ph.D. on Pattern Recognition and Intelligent Control from Tsinghua University in 1996. He chaired the IEEE Computer Society's Technical Committee on Services Computing from 2003 to 2011. He also chaired the Services Computing Professional Interest Community at IBM Research from 2004 to 2006. Dr. Zhang has served as the Editor-in-Chief of the International Journal of Web Services Research since 2003 and was the founding Editor-in-Chief of IEEE Transactions on Services Computing. He was elected as an IEEE Fellow in 2011, and in the same year won the Technical Achievement Award “for pioneering contributions to Application Design Techniques in Services Computing” from IEEE Computer Society. Dr. Zhang also chaired the 2013 IEEE International Congress on Big Data and the 2009 IEEE International Conference on Cloud Computing (CLOUD 2009).

About the Editor-in-Chief

Dr. Andrzej Goscinski is a full Professor in the School of Information Technology, Deakin University, Australia, where he directs research programs in clouds and cloud computing, parallel processing, virtualization, security, autonomic and service computing, and in general, distributed systems and applications. From January 1993 to December 2001, Dr. Goscinski completed tenure as the Head of School, and from 2004 he has led his research group to successfully concentrate their research on autonomic grids based on SOA, the abstraction of software and resources as a service, and cloud computing. A major achievement in the area of autonomic grids based on SOA was the development of the concept of a broker that led to its use in clouds. Furthermore, a major achievement in the area of the abstraction of software and resources as a service and cloud computing was the development of the Resource Via Web Services (RVWS) framework that contains service’s dynamic state and characteristics, and service publishing, selection and discovery; the contribution to level of cloud http://www.hipore.com/ijsc
abstraction in the form of CaaS (Cluster as a Service); comparative study of High Performance Computing clouds, and the development of H2D hybrid cloud. Currently, he concentrates his research on exposing HPC applications as services, publishing them to a broker, and executing them in a SaaS cloud by non-computing specialists. The results of this research have been published in high quality journals and conference proceedings. Dr. Goscinski serves as Associate Editor of IEEE Transactions on Service Computing; Associate Editor of Inderscience’s International Journal on Cloud Computing; member of the Editorial Board of Springer's Future Generation Computer Systems; and General, Program Chair, and Honorary Chair of IEEE Services and Cloud Conferences, and Distributed and Parallel Systems and Applications.

Dr. Jia Zhang is an Associate Professor at Carnegie Mellon University - Silicon Valley. Her recent research interests center on services computing, with a focus on scientific workflows, net-centric collaboration, Internet of Things, and big data management. She has co-authored one textbook titled "Services Computing" and has published over 130 refereed journal papers, book chapters, and conference papers. She is now an Associate Editor of IEEE Transactions on Services Computing (TSC) and of International Journal of Web Services Research (JWSR), and Editor-in-Chief of International Journal of Services Computing (IJSC). She earned her Ph.D. in computer science from the University of Illinoisat Chicago.
Call for Articles
IJSC Special Issue of Application-Driven Services Innovations

Services computing is a dynamic discipline. It has become a valuable resource and mechanism for the practitioners and researchers to explore the value of services in all kinds of business scenarios and scientific work. From industry perspective, IBM, SAP, Oracle, Google, Microsoft, Yahoo, and other leading software and internet service companies have also launched their own innovation initiatives around services computing.

The Services Transactions on Services Computing (IJSC) covers state-of-the-art technologies and best practices of Services Computing, as well as emerging standards and research topics which would define the future of Services Computing.

IJSC now launches a special issue which focuses on application-driven services innovations. The papers should generally have results from real world development, deployment, and experiences delivering SOA or solutions of web services. It should also provide information like "Lessons learned" or general advices gained from the experience of services computing. Other appropriate sections are general background on the solutions, overview of the solutions, and directions for future innovation and improvements of services computing.

Authors should submit papers (12 pages minimum, 24 papers maximum per paper) related to the following practical topics:
1. Architecture practice of services computing
2. Services management practice
3. Emerging services algorithms
4. Security application of services computing
5. Web services application practice
6. Micro-service practice
7. Quality of service for web services

Please note this special issue mainly considers papers from real-world practices. In addition, IJSC only considers extended versions of papers published in reputable related conferences. Sponsored by Services Society, the published IJSC papers will be made accessible for easy of citation and knowledge sharing, in addition to paper copies. All published IJSC papers will be promoted and recommended to potential authors of the future versions of related reputable conferences such as IEEE BigData Congress, ICWS, SCC, CLOUD, SERVICES and MS.

If you have any questions or queries on IJSC, please send email to IJSC AT ServicesSociety.org.

http://www.hipore.com/ijsc
USING SYNTACTIC AND SEMANTIC SIMILARITY OF WEB APIs TO
ESTIMATE PORTING EFFORT

Hiranya Jayathilaka, Alexander Pucher, Chandra Krintz, Rich Wolski
Department of Computer Science, UC Santa Barbara, CA, USA
{hiranya,pucher,ckrintz,rich}@cs.ucsb.edu

Abstract
Service Oriented Architecture (SOA) has altered the way programmers develop applications. Instead of using standalone libraries, programmers today often incorporate curated web services, accessed via well-defined interfaces (APIs), as modules in their applications. Web APIs, however, evolve rapidly, making it critical for developers to be able to compare APIs for similarity and estimate the workload associated with “porting” applications to use different or new APIs (or API versions). Unfortunately, today there is no simple automated mechanism for analyzing the similarity between web APIs and reasoning about the porting effort that will be necessary when the web APIs that an application uses change. To address this limitation, we describe an automated methodology for analyzing API similarity and quantifying the porting effort associated with the use of web APIs. Our approach defines a simple type system and a language with which API developers specify the syntactic and semantic features of APIs. We also define algorithms that transform the syntactic and semantic features of APIs into similarity and porting effort information. We evaluate our approach using both randomly generated and real-world APIs and show that our metric captures the relative difficulty that developers associate with porting an application from one API to another.

Keywords: Web services, Web APIs, Porting effort, Syntactic similarity, Semantic similarity, Axiomatic semantics

1. INTRODUCTION

Web services are widely used to implement Internet accessible applications. In this emerging development model, programmers combine extant network accessible services to create new applications. Developing applications out of curated web services improves programmer productivity over non-service-oriented methodologies by simplifying application assembly, testing, maintenance, and by improving the robustness of complex systems through the reuse of software and data components offered by providers “as-a-service”. By composing an application from existing services that encapsulate common yet complicated tasks, application developers are able to work at a higher level of abstraction, thereby saving valuable development and debugging time. Moreover, these composed applications leverage the stability and operational experience of their backend API providers.

A web service consists of one or more software components each with a well-defined, but, in terms of coding and implementation, separate application programming interface (API). The API is network-accessible and facilitates machine-to-machine interoperation. Separately, the web service “stack” is responsible for connecting each service implementation to the API code that exposes it to its users.

The growth in the popularity of this approach to application development has introduced several challenges for developers. In particular, because web-service-based applications decouple their service implementations from their APIs, the development and maintenance life cycles for APIs and service implementations are separated. As a result, APIs can and do change independently of the implementations they serve. In particular, new APIs (offering additional features as a superset) emerge frequently for existing services. Commercial service providers respond to competitive pressures by adding, modifying, deprecating, and retiring APIs regularly. Moreover, new APIs are introduced that are similar in functionality to existing APIs but that offer added functional and/or business advantages. Given such “API churn”, developers require new tools that help them reason about API similarity and the cost of migrating, i.e. porting, an application from one API to another to adapt to API changes.

Toward this end, we present a new approach that automates the process of evaluating the similarity between two APIs or API versions, and gives developers a way to estimate the “porting effort” required to update an application to use a new API or version. Without such support, developers have only their (error-prone) intuition or must speculatively execute a port to determine its suitability.

Our approach employs simple but formal mechanisms to analyze the similarity and compatibility of web APIs. In particular, we combine techniques that extract syntactic and semantic similarity from API operations. Our syntactic analysis precisely determines the input/output type compatibility between web APIs. Our semantic analysis captures the functional behavior of type-compatible API operations using syntactic structures. We then define a scoring metric that represents porting effort and can be used it to rank API alternatives.

To enable semantic analysis, we define a simple type system for web APIs and a semantic description language based on the popular Python programming language.
Developers use this type system and the semantic description language to document the important syntactic and semantic attributes of web APIs. Our syntactic similarity analysis compares the input and output data types of different APIs and determines if one API can be used to replace another at a syntactic level. The semantic similarity analysis makes use of the axiomatic semantics (i.e., preconditions and postconditions) of web API operations, to measure API similarity via an extended form of the Dice coefficient on the abstract syntax trees of semantic predicates, combined with Hoare’s consequence rule applied to API pairs. Use of axiomatic semantics allows service developers to easily document API semantics without delving into the internal implementation details of the web services. This specification language is familiar to many developers while facilitating simple static analysis.

We implement the proposed mechanisms and evaluate them using a number of popular APIs for social media login, airline itinerary search, and digital media video search. Our initial results indicate that developers can determine the similarity between web APIs and reason about the porting effort of migrating their applications to different web API versions and competitive implementations, without speculatively performing the porting. Our experimental results also show our approach to be efficient enough to be a practical part of the software engineering process used to develop service-composing applications. In the sections that follow, we detail our approach. We then describe the empirical evaluation of our algorithms, discuss the results, and conclude.

2. FROM API SIMILARITY TO PORTING EFFORT

We start with the hypothesis that application porting effort from one API to another is inversely proportional to the degree to which two APIs are similar. Two APIs are comparable in terms of porting effort if they are two different versions of the same API or expose same or similar services. API similarity can be syntactic, i.e., two APIs export operations with similar cardinality and data types for their inputs and outputs. Alternatively, similarity can be semantic, i.e., two APIs are similar in terms of the functionality and behavior of their syntactically similar operations. In this work, we propose mechanisms to analyze both syntactic and semantic similarity between web APIs.

The syntactic similarity between APIs provides a simple yet very effective means of establishing design-time or compile-time compatibility of different APIs. That is, if A and B are two syntactically similar APIs (i.e. they consume similar input data types and produce similar output data types), an application written using A can be easily modified and recompiled to use B. In other words, it results in low porting effort from API A to B and vice versa. Note that this notion of syntactic similarity is not too far from the traditional sense of API compatibility often discussed in programming languages and software engineering research. In fact, our algorithm for determining syntactic similarity among web APIs is heavily based on the typical type checking and verification methods used in the above-mentioned research areas.

While syntactic similarity is simple to analyze, when considering the porting effort among web APIs, it often results in insufficient or inconclusive information. To make a sound judgment regarding porting effort one must also consider the semantic similarity between the web APIs involved. This is because it is possible for two APIs to be syntactically identical, while having drastically different semantics. For example consider an API that takes two integers and returns their sum as the output. Now consider another API that also accepts two integers and returns their product as the output. These two APIs have identical input/output data types, but they accomplish very different tasks. Therefore while it is possible to easily rewrite and recompile an application based on the first API to use the latter API, the ported application will not work as expected due to the semantic difference between the two APIs.

To overcome this type of run-time inconsistencies, semantic similarity must be checked among the APIs that are involved in the port. A semantic similarity analysis would indicate very high porting effort between the two example APIs discussed above, while a purely syntactic similarity checker may determine the porting effort to be low which is misleading.

To estimate porting effort between web APIs, we propose a two-phase API similarity analysis. In the first phase APIs are subjected to a syntactic similarity check. This check results in a simple Yes/No answer indicating whether two APIs are compatible with each other or not. If this first step yields the APIs to be syntactically compatible, we proceed to the second phase, where we perform a semantic analysis on the APIs. Our semantic analysis results in a numeric value where higher values indicate higher porting effort (i.e. lesser similarity).

We next detail the syntactic and semantic similarity checking process. While both mechanisms answer the same question (i.e. whether two given web APIs are compatible) the two mechanisms can be studied, implemented and applied independently of each other. We find in this work that the best results are achieved when we apply the two mechanisms in combination.

3. SYNTACTIC SIMILARITY OF WEB APIs

In this section we overview our approach for establishing syntactic similarity between two web APIs. Syntactic similarity is primarily based on the inputs and outputs of API operations, their cardinality and data types. This is very similar to the notion of API compatibility commonly discussed in programming languages, compilers and software engineering research. Therefore the solution
Data: Source API S with operation set OP_S and Target API T with operation set OP_T

Result: A Boolean value and a set of matching operation pairs

M  ∅
for s ∈ OP_S do
  matched  FALSE
  for t ∈ OP_T do
    im  input_match(s.inputType, t.inputType)
    om  output_match(s.outputType, t.outputType)
    if im and om then
      OP_T  OP_T - { t }
      M  M ∪ { <s,t> } 
      matched  TRUE
      break
    end
  end
if not matched then
  return FALSE, ∅
end
return TRUE, M

Algorithm 1: Syntactic similarity checking algorithm

we propose is heavily inspired by this already existing research and widely used techniques.

Our algorithm takes two web API descriptions (source API and the target API) as the input and determines whether the calls to source API in an application can be syntactically replaced with calls to the target API. This basically amount to establishing that the target API supports all the operations of the source API. In other words, for each operation in source API, there should be a syntactically matching operation in the target API. The syntactic match (or syntactic compatibility) between two operations can be defined based on the following guidelines:

- Two operations accept identical or compatible input data types.
- Two operations produce identical or compatible output data types.

In order to be able to automatically check for these properties, we need a way to specify the type information regarding API operations in a machine-readable manner. This requires formulating a rich type system that can be used to document the type information regarding web APIs. Most real-world type systems can be used for this purpose. However, in order to maintain language and vendor neutrality, we present the following simple type system for web APIs. This type system has been inspired by several existing type systems used in various cross-language RPC frameworks (e.g. Apache Thrift), and API description languages (e.g. Swagger, WADL, JSON Schema). It is not tied to any specific programming language and therefore can be used to describe the web APIs implemented in any real-world language. Our type system consists of three categories of data types:

- Primitive types: boolean, byte, i16 (short), i32 (int), i64 (long), double, string, binary
- Container types: A list or a set of items, where all items are of the same type. A list is ordered and allows duplicates. Set is unordered and does not allow duplicates.
- Complex types: A type that consists of one or more attributes, where each attribute can be of any type.

This simple type system covers most data types encountered in real-world web APIs. It also enables defining high-level data types such as maps and other recursive data structures like lists of lists.

Web APIs often define input and output data fields as optional. To capture this information, we extend our type system with the ability to annotate objects and attributes as “required” or “optional”. We assume that the input API descriptions to our analysis contain this information alongside the type information. Other API description languages (e.g. Swagger, WSDL, WADL, JSON Schema) already provide support for such annotations.

Our algorithm for analyzing syntactic similarity between web APIs accepts a source API description and a target API description. For each operation in the source API description, it attempts to find a syntactically compatible operation in the target API. That is, for each source operation it attempts to find a target operation that accepts the same or fewer inputs, and produces the same or additional outputs. To define this notion formally, suppose I_S and O_S are the input and output types of the source API respectively. Similarly, assume that I_T and O_T are the input and output types of the target API. I_S and I_T are syntactically compatible, if I_T contains the same or less attributes as I_S. If I_T includes any attributes that are not present in I_S, they must be annotated as optional to maintain syntactic compatibility among the inputs. Similarly O_S and O_T are syntactically compatible, if O_T contains the same or more attributes as O_S. From an object-oriented programming perspective, I_S and I_T are syntactically compatible if I_T is a more general type (super type) of I_S. Similarly O_S and O_T are syntactically compatible if O_S is a more general type of O_T.

When comparing complex types for syntactic matches, the algorithm may encounter attributes, which in turn are of complex types (due to the recursive nature of the type system). In this case the algorithm must recursively compare the types of the child attributes. For example, assume a source operation, which has a complex input type C_S that contains an attribute A of type T_A. Now suppose there is a target operation, which has a complex input type C_T that also contains an attribute A, but of type T'_A. When comparing C_S against C_T, the algorithm must recursively compare the types T_A and T'_A for syntactic compatibility.
The algorithm iterates through the target API operations, looking for syntactic matches based on these guidelines. When it finds a matching target operation, the algorithm marks the operation, so that it is not matched with another source API operation. If the algorithm fails to locate a match for at least one source API operation, it returns FALSE to indicate syntactic incompatibility. It returns TRUE only if it can find matches for all source API operations. Algorithm 1 further describes this analysis. The procedures input_match and output_match are recursive functions that take two types as the input, and check for their syntactic compatibility based on the rules described earlier.

Based on the additional information available in the input API descriptions, we can make the syntactic analysis more sophisticated and accurate. For example, in addition to simply comparing the input/output data types, we can also compare the HTTP methods of operations, payload mime types and status codes returned by the APIs. This way, a source operation that consumes a JSON payload sent as a HTTP POST request and produces HTTP 201 responses, will only be matched against a target operation, which also consumes JSON payloads sent as HTTP POST requests and produces HTTP 201 responses in return. Most existing API description languages already capture this additional information regarding API operations, and hence they can easily be included in a syntactic similarity analysis.

4. SEMANTIC SIMILARITY OF WEB API

To define a metric for application porting effort from one API (source API) to another (target API) using the semantics of their operations, we require mechanisms

- with which API developers specify the semantics of API operations
- that automate the consumption and analysis of specified API semantics, and
- that use the output from the analysis to construct a measure of porting effort for a pair of APIs

To define these mechanisms and the overarching metric, we leverage and assemble extant research advances in a simple, yet new way that enables developers to estimate and rank the effort associated with porting their application to a different version of a web API or to an alternative implementation of an API. For simplicity of discussion, we assume that a pair of APIs under consideration has a single, syntactically matching operation. That is, in what follows, we will examine the ability to quantify similarity between individual API operations. As part of our future work we plan to extend the methodology to consider multiple operations in pairs of APIs.

4.1 Specifying API Semantics

The first mechanism of our approach is a specification language that developers can use to document the semantics of the operations in their web APIs. Our goal is to define a language that is simple, familiar, and intuitive to use that, at the same time, enables developers to specify the meaning of an API in a way that is amenable to efficient static analysis for semantic similarity. Toward this end, we leverage popular programming language syntax and tooling, and by previous works such as JML and Spec# that document program semantics (behavioral interface specifications) using programming language syntax. This latter research and that of others shows that using the syntax of familiar and popular programming languages to document API semantics facilitates programmer creation and editing of semantic specifications.

We restrict the Python language in a number of ways to facilitate analysis and to simplify the specification process by API developers. Our language only accepts single-lined Python statements that are free of side effects. We disallow side effects to preclude the consideration of internal service state. We also disallow conditionals, loops, try-catch blocks, class definitions, and function definitions.

Developers use this language to describe the behavior of API operations using axiomatic semantics -- preconditions that hold prior to invoking the operation and postconditions that hold after the operation executes. We leverage axiomatic semantics as a first step toward describing and analyzing API operations in a way that reflects porting effort. We plan to consider other successful approaches to describing the function and behavior of API operations as part of future work.

Developers refer API request parameters and response parameters using the built-in logical variables input and output, respectively. For example, for an operation that takes two positive numbers and responds with their sum, the preconditions can be documented using the statements input.x > 0 and input.y > 0; the postconditions can be documented as output.sum == input.x + input.y. These logical variables have been inspired by Hoare logic and separation logic to differentiate precondition values from postcondition values. The use of logical variables also enables expressing postconditions relative to preconditions, that is, postconditions can refer to the pre-state (request state) of an operation.

We do not allow invoking arbitrary functions using our language. This includes the built-in functions of Python as well as any class-level functions that can be invoked as object methods. However, we do support a number of useful predefined, side-effect-free, functions (that we have defined) when invoked as built-in functions (as opposed to object methods). We currently support the functions len, implies, forall, exists, matches, datebefore, and dateformat. We illustrate the use of a subset of our built-ins using simple examples below. Our language and built-ins are easily extended if and when more expressive power is required.

http://www.hipore.com/ijsc/
Password input must be at least 6 characters long:
  o \( \text{len(input.password)} \geq 6 \)

All entries in the input list named scores must be within the range \([0,100]\):
  o \( \forall \text{entry}, \text{input.scores}, 0 \leq \text{entry} \leq 100 \)

The format of the publishedDate output field is yyyy-MM-dd:
  o dateformat(output.publishDate, `yyyy-MM-dd')

If the country output field is set to US, the currency output field will be set to USD:
  o implies(input.country == `US', output.currency = `USD')

Note that most of the above functions are not part of the standard Python programming language. We have added them in our API description language as native constructs. The following examples illustrate how some of the above functions can be used to document API preconditions and postconditions. As seen from the above examples, our Python-based syntax coupled with the built-in functions, can be used to document even the most complex of the API semantics. The language can be easily understood by human developers, and can be easily processed by programs using a simple language parser. New built-in functions can be introduced to extend the language, and enhance its expressive power.

4.2 Comparing API Operations Pairwise

We next determine a similarity “score” by comparing the preconditions and postconditions of individual API operations. Throughout the remainder of this paper, we refer to the specified preconditions and postconditions of an API simply as semantic predicates. We represent semantic predicates as abstract syntax trees (ASTs).

To compare a pair of matching API operations, we compute a tree similarity metric on their ASTs. To enable this, we employ a technique that is widely used for software plagiarism detection and source code evolution analysis, called the Dice coefficient. The Dice coefficient has been shown in this past work to accurately extract the semantic similarity of two code fragments. Using the Dice coefficient, we treat each AST as a set of nodes over which we compute set similarity. Specifically, if \( P_1 \) and \( P_2 \) are two semantic predicates whose ASTs are \( T_1 \) and \( T_2 \) respectively, we compute the degree of similarity between the predicates \( P_1 \) and \( P_2 \) by computing the Dice coefficient on \( T_1 \) and \( T_2 \) as follows.

\[
\text{Similarity}(P_1, P_2) = \text{Dice}(T_1, T_2) \\
\text{Dice}(T_1, T_2) = \frac{2C}{2C + L + R}
\]

C is the number of nodes common to both \( T_1 \) and \( T_2 \). L is the number of nodes unique to \( T_1 \) and R is the number of nodes unique to \( T_2 \). This approach enables us to obtain a similarity value between 0 and 1 for any two given semantic predicates, where 0 indicates a total mismatch and 1 indicates a perfect match.

We also apply a trivial transformation on the semantic predicates when performing semantic comparison that breaks disjunctive and conjunctive predicates into their constituent predicates. This enables our mechanism to handle situations where the same set of predicates has been expressed in two APIs, but in slightly different formats.

Notice that the amount of work necessary to port from one API to another is affected by the number of predicates in each. In particular, the effort to port from a source API with fewer preconditions than the target API is more difficult than porting in the reverse direction.

To illustrate this asymmetry, let \( M \) and \( N \) be two web APIs where \( N \) has more preconditions than \( M \). It is more difficult to port from \( M \) to \( N \) than from \( N \) to \( M \). More

---

Data: Source API \( S \) with predicate sets \( S_{pre}, S_{post} \) and Target API \( T \) with predicate sets \( T_{pre}, T_{post} \)

Result: Porting effort

\[
\text{M}_{pre} \leftarrow \emptyset, \text{M}_{post} \leftarrow \emptyset \\
\text{P}_{eff1} \leftarrow 0, \text{P}_{eff2} \leftarrow 0 \\
\text{Temp1} \leftarrow \text{EmptyMap}, \text{Temp2} \leftarrow \text{EmptyMap} \\
\text{for } <x,y> \in (S_{pre} \times T_{pre}) \text{ do} \\
\quad \text{map_store(Temp1, } <x,y>, \text{ Sim(<x,y>))} \\
\text{end} \\
\text{while unmarked(S_{pre}) and unmarked(T_{pre}) do} \\
\quad <x,y>, D_x \leftarrow \text{map_get_max(Temp1)} \\
\quad \text{mark}(S_{pre}, x), \text{mark}(T_{pre}, y) \\
\quad \text{map_remove(Temp1, } <x,y>) \\
\quad \text{M}_{pre} \leftarrow \text{M}_{pre} \cup \{<x,y>\} \\
\quad \text{P}_{eff1} \leftarrow \text{P}_{eff1} + (1 - D) \\
\text{end} \\
\text{P}_{eff1} \leftarrow \text{P}_{eff1} + |T_{pre}| - |M_{pre}| \\
\text{for } <x,y> \in (S_{post} \times T_{post}) \text{ do} \\
\quad \text{map_store(Temp2, } <x,y>, \text{ Sim(<x,y>))} \\
\text{end} \\
\text{while unmarked(S_{post}) and unmarked(T_{post}) do} \\
\quad <x,y>, D_y \leftarrow \text{map_get_max(Temp2)} \\
\quad \text{mark}(S_{post}, x), \text{mark}(T_{post}, y) \\
\quad \text{map_remove(Temp2, } <x,y>) \\
\quad \text{M}_{post} \leftarrow \text{M}_{post} \cup \{<x,y>\} \\
\quad \text{P}_{eff2} \leftarrow \text{P}_{eff2} + (1 - D) \\
\text{end} \\
\text{P}_{eff2} \leftarrow \text{P}_{eff2} + |S_{post}| - |M_{post}| \\
\text{return } \text{P}_{eff1} + \text{P}_{eff2}
\]

Algorithm 2: Porting effort evaluation algorithm

http://www.hipore.com/ijsc/
preconditions imply that N’s input set is more restricted than M. Therefore it cannot support all the inputs that M does. Hence some extra effort has to be put in by the developer to make sure that the application doesn’t pass an unsupported input value to API N. However, by the same argument, porting an application from N to M should be easier. Since M’s input set is less restricted than N, the developer doesn’t have to do any extra work in this case.

Notice also that a similar asymmetry exists with respect to postconditions. If an application is to be ported from API S to API T and if T has more postconditions than S, then porting S to T is easier than the other way around. More postconditions help further restrict the output of API T. In other words, T may not produce an output that S doesn’t. Therefore the application should be able to handle all the outputs generated by T, without having to make any code changes. Also, porting from API T to S becomes more difficult, since S might produce an output that T doesn’t.

4.3 Quantifying Porting Effort of Operations

Using the mechanism described in the previous section, we construct a measure of application porting effort using the semantic similarity of two APIs. Suppose S is a source API with the precondition set \( S_{\text{pre}} \) and the postcondition set \( S_{\text{post}} \). Suppose T is a target API with the precondition set \( T_{\text{pre}} \) and the postcondition set \( T_{\text{post}} \). To compute the porting effort from S to T, we first compare each member in \( S_{\text{pre}} \) against each member in \( T_{\text{pre}} \). That is, we calculate the similarity (Dice coefficient) of each predicate pair in \( S_{\text{pre}} \times T_{\text{pre}} \). Then we choose the pairs with the highest similarity, and match each member in \( S_{\text{pre}} \) to a member in \( T_{\text{pre}} \). In other words, for each predicate \( x \in S_{\text{pre}} \) we assign a predicate \( y \in T_{\text{pre}} \) such that the similarity of \( <x,y> \) is greater than the similarity of any \( <x,z> \) where \( z \in T_{\text{pre}} \) and \( y \neq z \). Matched pairs are put into a new set \( M_{\text{pre}} \). We also make sure that no member in \( S_{\text{pre}} \) or \( T_{\text{pre}} \) is matched to multiple counterparts. That is, whenever we insert a pair \( <x,y> \) into \( M_{\text{pre}} \), we mark \( x \) in \( S_{\text{pre}} \) and \( y \) in \( T_{\text{pre}} \) so that they cannot be considered for a match again. This way each member in \( S_{\text{pre}} \) can be matched to a unique member in \( T_{\text{pre}} \), as long as \( |S_{\text{pre}}| \leq |T_{\text{pre}}| \). But if \( |S_{\text{pre}}| > |T_{\text{pre}}| \) some members of \( S_{\text{pre}} \) will remain unmatched.

We translate the predicate assignments into a porting effort score by computing \((1 - D_i)\) where \( D_i \) is the similarity of the pair \( i \in M_{\text{pre}} \). We add these values up to obtain an initial porting effort score \( P_{\text{eff1}} \). Then we consider the remaining unmatched (unmarked) predicates in \( S_{\text{pre}} \) and \( T_{\text{pre}} \). Recall that porting to an API with more preconditions is more difficult than in the reverse direction. To reflect this asymmetry in our methodology, we increase \( P_{\text{eff1}} \) by 1 for each unmatched predicate in \( T_{\text{pre}} \). Unmatched predicates in \( S_{\text{pre}} \) are ignored. Therefore, we have:

\[
P_{\text{eff1}}(S, T) = \sum_{i \in S_{\text{pre}}} (1 - D_i) + |T_{\text{pre}}| - |M_{\text{pre}}|
\]

We perform a similar computation for postconditions using the sets \( S_{\text{post}} \) and \( T_{\text{post}} \). We compute the similarity of the members of \( S_{\text{post}} \times T_{\text{post}} \) and pick the pairs with the highest similarity to initialize a matching set \( M_{\text{post}} \). As a postcondition pair \( <x,y> \) inserted to \( M_{\text{post}} \), we mark \( x \) in \( S_{\text{post}} \) and \( y \) in \( T_{\text{post}} \) to ensure that no predicate is matched multiple times. Then for each pair \( j \in M_{\text{post}} \) we compute \((1 - D_i)\) where \( D_i \) is the similarity of the pair \( j \), and add these values up to obtain the porting effort score \( P_{\text{eff2}} \). We further penalize the porting effort by increasing \( P_{\text{eff2}} \) by 1 for each unmatched (unmatched) predicate in \( S_{\text{post}} \). This adjustment accounts for the greater difficulty associated with porting from an API with more postconditions to one with fewer postconditions.

\[
P_{\text{eff2}}(S, T) = \sum_{j \in M_{\text{post}}} (1 - D_j) - |S_{\text{post}}| - |M_{\text{post}}|
\]

We calculate the final porting effort score by combining the values from previous computations. If \( P_{\text{eff}}(S, T) \) is the porting effort from API S to API T, we have:

\[
P_{\text{eff}}(S, T) = P_{\text{eff1}}(S, T) + P_{\text{eff2}}(S, T)
\]

Algorithm 2 further illustrates our porting effort evaluation method. Temp1 and Temp2 are map data structures that support storing key-value pairs. The algorithm makes use of following named procedures:

- map_store(map, key, value) - Stores the given key-value pair in the map.
- map_get_max(map) - Returns the key-value pair with the largest value in the map.
- map_remove(map, key) - Removes the entry with the specified key from the map.
- mark(set, element) - Marks the specified element in the set.
- unmarked(set) - Returns TRUE if the set contains at least one unmarked element. Otherwise returns FALSE.
- Sim(<x,y>) - Returns the similarity (Dice coefficient) of the predicate pair \(<x,y>\).

5. TWO-PHASE API SIMILARITY ANALYSIS

In this section we combine our syntactic analysis and semantic analysis into a single algorithm. The inputs to the algorithm are two web API descriptions (the source API and the target API), documented using our type system and the Python-based semantic description language. Algorithm outputs a sequence of matching (i.e. syntactically compatible) operation pairs and the porting effort value for each pair. If the algorithm fails to detect any syntactically compatible operation pairs between the source and target API, it simply returns an empty set.

http://www.hipore.com/ijsc/
The algorithm first performs syntactic similarity analysis on pairs of operations. Each pair consists of one operation from the source API, and one from the target API. We attempt to match each source API operation with a syntactically compatible target API operation. The algorithm returns the empty set and halts if it cannot find a matching target operation for at least one source API operation. The algorithm ensures that each target API operation is matched to at most one source API operation.

If this initial phase of syntactic analysis succeeds in matching all source API operations with target API operations, the algorithm proceeds to the second phase. Here the algorithm performs a semantic analysis on each of the matched operation pairs. Final output of the algorithm is a list of matching operation pairs and their corresponding porting effort values. If the algorithm returns the empty set (in first phase), it implies that a straightforward port between the given source and target APIs is not possible (i.e. at least one of the required operations are not supported by the target API). If the algorithm returns a list of matching operations, the associated porting effort values can be used to estimate the difficulty of the port in practice.

Algorithm 3 illustrates the outline of our two-phase API similarity analysis method. The procedures “syntactic_similarity” and “semantic_similarity” in the listing are functions that invoke algorithm 1 and algorithm 2 respectively. The procedure “define_api” is a helper method that defines a temporary API specification from the operation provided as input. This is there simply because we have defined algorithm 2 to accept two complete API specifications as the input. In a real-world implementation this can be simplified or even avoided if necessary.

### 6. Prototype Implementation

We implement the proposed syntactic similarity analysis and the semantic similarity analysis as a command-line tool. This tool is programmed in Python and in total consists of around 750 lines of code. It takes as input two API descriptions documented using an extended form of Swagger. Swagger is a popular JSON-based description language that syntactically describes REST APIs. It uses a type system very similar to the one described in section 3, and also captures individual operation names, HTTP methods, media types of message payloads and error codes. We extend the base Swagger description language by introducing two new JSON attributes to the operation description. These attributes are named “requires” and “ensures” (inspired by JML). Each attribute points to a list of semantic predicates written using our Python-based semantic description language. The “requires” attribute holds the preconditions of the operation, and the “ensures” attribute holds the postconditions. This extension results in a more complete API description that consists of both type information (for syntactic similarity checking) and axiomatic semantics (for semantic similarity checking).

Our decision to base our prototype on the Swagger API description language has been motivated by several reasons. These include simplicity, openness of the standard, widespread adoption in the industry, existence of many tools and libraries to process Swagger descriptions and existence of tools to auto-generate Swagger descriptions from web service codes.

We have kept our prototype very simple and lightweight. In its present state, it does not make use of any third party libraries except for the standard Python modules. Swagger specifications are read from the file system and parsed as JSON strings using Python’s native JSON support. Semantic predicates are parsed into their AST representations using Python’s built-in “ast” module. This greatly simplifies the implementation, and prevents us from having to write our own grammar rules or parser to process semantic predicates.

#### 6.1 Auto-generating API Specifications

In this section we briefly discuss the issue of auto-generating API descriptions with type information and semantic predicates, so they can be used for the type of analyses described in our work. We believe that the ability to auto-generate details API specifications is crucial for this type of automated analyses and tools to be widely adopted and deployed in the industry. Handcrafting specifications for complex web APIs takes time, can be error prone and can result in various software maintenance complexities in the long run.

As a part of our research, we have implemented tools that can auto-generate Swagger API descriptions from the web services coded in Java (JAX-RS) and Python. The auto-generated specifications list the operations of the APIs, along with their HTTP methods, status codes, mime types and input/output data types. Swagger uses a type system...
very similar to the one discussed in section 3, which is serialized into JSON Schema. In case of Java web services, we have implemented a Maven plug-in that gets activated at the compile-time of the source code, which performs static analysis on the code to generate the necessary Swagger API descriptions. It extracts the necessary metadata out of method signatures and JAX-RS annotations and Javadoc comments present in the code. In case of Python (which is not a compiled language), we provide a separate command-line tool that needs to be invoked manually to parse the source and generate the API specifications. This tool also extracts the required metadata from Python method signatures, decorators and docstrings available in the code.

Our tools currently do not facilitate generating API specifications with semantic information. We have left this feature for future work. We intend to utilize the techniques popularized by frameworks such as JML and PyContracts to extract the required axiomatic semantic predicates from source code into the API specifications. That is, the developers will be required to document their source code with the proper axiomatic semantics (using comments and annotations), and the API specification generators will pick up this information from the code. The design by contract research corpus already describes mechanisms that can be used to automatically check and enforce those semantic constraints at run-time, which will ensure that the web service implementations never stray away from their documented semantic contracts.

7. Experimental Results

We have developed our prototype so as to be able to separately evaluate each phase of the analysis. We first consider syntactic similarity analysis and then evaluate semantic analysis in detail. For the latter, we consider randomly generated API specifications to study various characteristics of our API porting effort metric. We then consider real-world APIs and developer-perceived porting effort, and evaluate the overhead of our approach.

7.1 Syntactic Similarity Results

To evaluate the effectiveness of our syntactic similarity analysis, we take the Swagger specification of an existing test API, and create multiple modified versions of it. Each modified version demonstrates a possible way the input and output data types of an API operation can change in real-world API deployments. Then we run our syntactic similarity analysis algorithm on the original API specification and each of its modified versions, and record the output of the algorithm. In addition to the simple TRUE/FALSE output of the algorithm, our prototype implementation also gives a textual description of the changes it detects between compared API specifications. We record these results in Table I.

Our experimental results show that the proposed syntactic similarity analysis is capable of detecting all possible ways data types of an API operation can change (i.e. addition, removal and modification of type attributes). Further, our prototype is capable of pinpointing the exact differences between input/output types of APIs, when there are incompatibilities among them.

Table I. Syntactic similarity analysis results.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Expected Result</th>
<th>Actual Result</th>
<th>Generated Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adding a new optional input parameter</td>
<td>TRUE</td>
<td>TRUE</td>
<td>None</td>
</tr>
<tr>
<td>Adding a new required input parameter</td>
<td>FALSE</td>
<td>FALSE</td>
<td>Required input parameter introduced in new API</td>
</tr>
<tr>
<td>Removing an input parameter</td>
<td>TRUE</td>
<td>TRUE</td>
<td>None</td>
</tr>
<tr>
<td>Renaming a required input parameter</td>
<td>FALSE</td>
<td>FALSE</td>
<td>Required input parameter introduced in new API</td>
</tr>
<tr>
<td>Renaming an optional input parameter</td>
<td>TRUE</td>
<td>TRUE</td>
<td>None</td>
</tr>
<tr>
<td>Adding a new optional output parameter</td>
<td>TRUE</td>
<td>TRUE</td>
<td>None</td>
</tr>
<tr>
<td>Adding a new required output parameter</td>
<td>TRUE</td>
<td>TRUE</td>
<td>None</td>
</tr>
<tr>
<td>Removing an output parameter</td>
<td>FALSE</td>
<td>FALSE</td>
<td>No match found for output field</td>
</tr>
<tr>
<td>Renaming a required output parameter</td>
<td>FALSE</td>
<td>FALSE</td>
<td>No match found for output field</td>
</tr>
<tr>
<td>Renaming an optional output parameter</td>
<td>FALSE</td>
<td>FALSE</td>
<td>No match found for output field</td>
</tr>
</tbody>
</table>

Figure 1: Average execution time of the syntactic analysis
Next, we evaluate the performance of syntactic analysis. We handcraft a series of API specifications, with different attribute (parameter) counts per input/output type, and different levels of recursion (nesting of types within a type). We compare each specification against itself using our algorithm 1000 times, and calculate the average execution time of a single run of the algorithm. Figure 1 depicts the results of this experiment.

The data shows that our syntactic similarity analysis scales linearly with the number of attributes available in data types. Also note that the y-axis of Figure 1 is in milliseconds, which implies negligibly small overhead (<5ms) even in the worst case under consideration (50 attributes per type, with 3 levels of nesting).

7.2 Randomly Generated APIs

In our next experiment, we randomly generate a population of 100 API specifications. Each specification has a single operation. We semantically compare each API against all others in the population and compute the porting effort between them. We repeat this experiment using different numbers of semantic predicates. We randomly generate the API specifications with 10, 20 and 50 semantic predicates. Our goal with this experiment is to understand how our measure of porting effort changes under these scenarios (e.g. to determine the sensitivity of the mechanism to supplied parameters).

Figure 2 shows the cumulative distribution functions (CDFs) of the computed porting effort as a function of the number of predicates per single API operation. A porting effort value of 0 indicates no porting effort. The data shows that the porting effort between API operations increases with the number of semantic predicates. For example, the maximum porting effort observed in APIs with 10 semantic predicates is 17.4. This goes up to 30.1 when the number of predicates is increased to 20. It further increases up to 44.9 when the semantic predicates count is set to 50. Also, when considering the CDFs of the porting effort, 50% of the API operation pairs have 4.3 or less porting effort in the population with 10 semantic predicates. In the population with 20 semantic predicates, 50% of the APIs have 7.1 or less porting effort. In the population with 50 semantic predicates, this limit further increases up to 12.9. This is inline with our experience in which, as the number of semantic predicates increases, the API consumer is forced to adhere to additional restrictions. As such, when porting among different web API operations, the developer has to take more constraints into account and must write more code to reconcile the differences. This results in increased porting effort. Our experimental results suggest that our porting effort metric captures this phenomenon.

It is also interesting to note that our porting effort values are not bounded by any upper limit. The porting effort could be arbitrarily large depending on the number and the complexity of the semantic predicates. We believe that this property of the metric reflects current practice. That is, it is always possible to find or create two new APIs E and F, such that the effort it takes to port an application from E to F is greater than any previously known upper bound. Our porting effort evaluation mechanism captures this property.

7.3 Publicly Available, Real-World APIs

We next investigate the efficacy of our approach using popular, publicly available web APIs. We list these APIs below. To evaluate our porting effort metric, we have augmented the APIs with semantic specifications manually. To enable this, we carefully analyze the API documentation and examples related to each of these web APIs. Specifically, we identify an important operation from each API set that was present across the set and specify its pre/postconditions using our specification language. Thus, these results pertain the similarity between an individual API operation that is common to all APIs in a set (either social media, airline services, or digital media).

- Social media login APIs: Facebook, Google, LinkedIn, Twitter, Yahoo, Hi5, Amazon
• Airline itinerary search APIs: American Airlines, British Airways, Cathay Pacific, Delta Airlines, Emirates, Etihad, Singapore Airlines, United Airlines, Virgin America
• Digital media video search APIs: Youtube, iTunes, MovieDB, RottenTomatoes, Vimeo

We then compute the porting effort among each pair of APIs within each of the above three categories. We present the CDFs of the results in Figure 3.

The data shows that a fairly large proportion of the API pairs have a low porting effort. For instance, in all three populations (social media, airlines and video search), 50% of the pairs have a porting effort of 3.3 or less, a characteristic not present in the data obtained from the randomly generated APIs. This is because, unlike in the randomly generated populations where most APIs are completely unrelated to each other, in real world API populations most APIs can and do have commonalities. For instance, most social media login APIs have similar constraints on username and password. Most airline APIs have similar requirements with respect to specifying departure and arrival cities, travel dates and the number of passengers. Most video search APIs also exhibits similar constraints, in that most APIs at least accept simple text queries to perform keyword-based search. These similarities simplify application porting.

The CDFs of the social media APIs and the airline APIs follow relatively similar trends. However, the CDF of the video search APIs deviates from the other two and reaches a maximum porting effort value close to 35. A closer look at the API specifications showed that social media APIs and the airline APIs are similar in terms of their average semantic predicate count (8.1 and 9.3 respectively). For the video search APIs, the average predicate count is as high as 15.6 thus resulting in an increased porting effort among them. Also, some of the video-search APIs have a large number of semantic predicates compared to the others. For instance, Youtube search API has 28 semantic predicates, and the iTunes search API has 30 semantic predicates. Therefore ports that involve these APIs tend to be much more complicated than the others.

7.4 Categorizing API Porting Difficulty

Given this efficacy (particularly for the real-world APIs), we can determine categories of difficulty. That is, we can use the methodology to “cluster” API ports into groups that can be ranked in terms of difficulty (e.g. is a port “easy” or “hard”?)

To investigate this hypothesis we use k-means clustering to classify the results into two groups (i.e. k = 2). Figure 4 shows, for each sample set, the ratio of the variance explained by the categorization to the total variance in the set. Typically, this analysis shows an “elbow” in the curve corresponding to the point where further categorization adds little explanatory power. In our study, that point of diminishing returns appears at k = 2.

Thus, for these API operations, it appears that our methodology should be able to divide pairwise porting effort into two categories: “easy” and “difficult”. We then asked two of our lab members (lets call them D1 and D2) conversant with web services but not otherwise associated with this project to categorize the porting difficulty of a subset of the porting possibilities in each set as either “easy” or “difficult”.

We gave these developers three sample sets, each consisting of 5 API specifications, randomly chosen from the above three categories (social media, airlines and video search). We then asked each developer to analyze the API specifications pairwise, and classify all possible pairs into two groups -- easy and difficult -- depending on the potential complexity of porting an application from one API to another. We also computed the porting effort between these web APIs using our own prototype, and used k-means clustering to classify the results into two groups (i.e. k = 2).

Figure 5 shows the percentage accuracy of the classifications computed using our formal mechanism with

http://www.hipore.com/ijsc/
respect to the classifications provided by developers D1 and D2 respectively.

We compute the percentage accuracy as the ratio of the number of entries classified as the same (i.e. agreement between the developer and the methodology) to the total sample size. In terms of a simple categorization, the agreement is good. Indeed, developer D2 and the methodology obtained the same classification (100% accuracy) for the social media API operation.

7.5 Overhead of Semantic Analysis

Finally, we the time overhead associated with computing porting effort using our mechanism. We employ our randomly generated set of 100 API specifications and compute the porting effort between each pair of APIs. We measure the time elapsed for all steps and then compute the average time per API pair. We repeat the experiment, varying the total number of semantic predicates in each API specification. We report the average times that we observe in these experiments in Figure 6.

For web APIs with 10 semantic predicates, our evaluation method takes less than 10ms. This increases up to 200ms when the predicate count is increased to 50. This increase in execution time is due to the pairwise AST comparison operations performed by our algorithm. That is, when computing the porting effort between two APIs, our prototype compares each preCondition of the source API against each preCondition of the target API. In the same fashion, our prototype compares each postCondition of the source API against each postCondition of the target API. Therefore the number of AST comparisons performed is polynomial in the number of semantic predicates. Hence the average execution time of our algorithm increases polynomially with the increase in semantic predicates. However, for web APIs with 10 semantic predicates, the average execution time is below 10ms and for web APIs with 20 semantic predicates, the average execution time is well below 50ms. Since most of the real world web APIs that we have studied to date have a small number of predicates (the max was 30), our approach is not likely to impose a significant time overhead on the development process for applications. If required, the algorithm can be easily parallelized by running the pairwise AST comparisons in parallel to reduce the overhead further.

Compared to the execution time of the syntactic similarity analysis, however, the semantic similarity analysis takes much longer to complete. This indicates that in our two-phased API similarity analysis algorithm, the semantic similarity analysis component is the more expensive and critical element in terms of time complexity.

Overall, our porting effort evaluation method produces useful results with a high level of accuracy. The method is efficient, and can be easily applied to real world web APIs. The Python-based syntax simplifies documentation and publication of API semantics (relative to semantic ontologies, state machines, and formal logic) by API providers. If an API provider fails to publish API semantics in our language, API consumers (developers) can easily create API specifications on their own by converting the semantics of API operations described in the API documentation into Python code.

8. Related Work

This paper is an extension of our initial investigations into semantic analysis of web APIs. This work, in general, builds upon and extends research from a number of other areas in computer science. These areas include programming language and web service semantics, analysis and verification.

Static type checking techniques have been in widespread use for decades and make up one of the corner stones of programming languages research. We employ some very traditional and basic type checking mechanisms to implement our syntactic similarity analysis. The proposed input/output type comparison rules have strong roots in existing type checking techniques and object-oriented programming. Our type system has been inspired by a number of other type systems used in cross-language RPC frameworks (e.g. Apache Thrift, Google Protocol Buffers) and syntactic API description languages (e.g. Swagger, JSON Schema, WADL). Like the type systems of cross-language RPC systems, our type system is also not tied to any specific programming language. It facilitates specifying optional and required data fields, much like how most API description languages support annotating data fields as either required or optional.

Our approach of using axiomatic semantics to describe web APIs is rooted in the work of Floyd and Hoare. Floyd modeled computer programs as digraphs where vertices represent program statements and edges represent control flow. Predicates representing correctness conditions are attached to the edges. Hoare introduced the notion of Hoare triples and constructed a formalism for reasoning about program correctness using them. A Hoare triple is a logical
construct of the form $P(C)Q$ where $C$ is a command (an operation) in a program, $P$ is the set of preconditions of $C$ and $Q$ is the set of postconditions of $C$. We adapted this formalism into our work where we reason about web services by describing their operations along with the respective preconditions and postconditions. Hoare's seminal work on using axiomatic semantics to reason about program correctness excludes side effects and arbitrary procedure calls. In this work, we follow the same approach for semantic predicate description language to facilitate low complexity and thus fast evaluation of API porting effort.

Several researchers have been successful in using axiomatic semantics to reason about the correctness and behavior of software constructs. Hoare himself, along with Wirth showed how axiomatic semantics can be used to describe Pascal programs. Fikes and McGuiness used axiomatic semantics to describe RDF data models. Gegg-Harrison et al introduced ProVIDE, a software development tool that allows the user to establish program correctness via specifying postconditions and then generating the corresponding preconditions. Black used axiomatic semantics to verify the behavior of a secure web server.

Our guidelines for comparing web API semantics are loosely based on Hoare's rule of consequence. The rule of consequence states that if $P(C)Q$ and $P(C')Q'$ are two Hoare triples such that $P \rightarrow P'$ and $Q' \rightarrow Q$, then the command $C'$ can be used in any context where the command $C$ can be used. This is because $C'$ has more permissive preconditions and more restrictive postconditions compared to $C$. We follow a similar rule when comparing web APIs with unequal number of preconditions or postconditions. Naumann and Olderog have made similar arguments.

The use of programming language syntax for expressing program semantics and contracts is a widely used concept. JML uses two primary annotations (requires and ensures) to document the preconditions and postconditions of Java methods using Java syntax. Spec# provides similar functionality for the C# language. SPARK language has built-in contract documentation features, where contracts are encoded in the source code as Ada comments. These technologies enable automated semantic or contracts mostly for verification purposes. That is, they verify whether the program adheres to the given contract at the runtime. We use the documented semantics at the development time to reason about web service semantics and porting effort by applying static analysis methods.

The use of AST representations to compare programs and reason about them is also well researched. Our approach is heavily based on the work of Baxter et al, where they used AST comparison methods for detecting program clones. Baxter et al introduced the notion of syntactic similarity (based on the Dice coefficient), as opposed to exact matches, as a more practical means of finding program segments with similar functionality and behavior. Cui et al showed how to use AST comparison methods for source code plagiarism detection. They showed that AST comparison based methods are capable of finding a wide range of similarities between different programs. Hashimoto and Mori augmented AST comparison methods with heuristics-driven techniques so that they can be used to efficiently analyze the differences between programs written in a wide range of programming languages. Neamtiu et al used AST comparison methods to track down and analyze how a program code base has evolved over time.

Bianchini et al introduced the notion of semantics-enabled web API selection patterns. One of the selection patterns they discuss is the substitution pattern, which aims at finding a web API that can be used to substitute another API (i.e. porting). They presented a formalism to model and quantify this selection pattern based on semantic ontologies. However, constructing comprehensive semantic ontologies requires a lot of time and manual effort, and therefore such techniques are difficult to apply in practice. Also they mainly focus on semantically annotating the input/output data elements of web APIs, whereas we look at the functional and behavioral traits of the web APIs.

There is a large body of work that uses techniques like process models, state machine models and logic to reason about web service behavior. However these formalisms are aimed at addressing the issues such as discovery, monitoring and verification. Our work deviates from these formal methods, in the sense we attempt to reason about developer experience of different web services, in the sense how much effort a developer has to put in to port an application from one web API to another.

9. Conclusions

Increasingly, web, mobile, and cloud developers integrate publicly available web services, exposed via well-defined APIs, into their Internet accessible applications. Doing so simplifies and expedites software development, testing, deployment, and management of these applications. Despite these benefits that arise from decoupling of APIs from the implementations they serve, the web service model has also introduced a key challenge for developers: API churn -- constant API evolution (versioning) and emergence of alternative, competitive implementations for the same API. As a result, it is critical that developers be able to efficiently analyze the similarity between APIs and reason about the work required to migrate their applications from one API (or API version) to another.

In this paper, we investigate a new methodology for automatically analyzing API similarity and quantifying application porting effort. Our approach defines a basic recursive type system and a simple language based on Python with which API developers document the syntactic and semantic aspects of API operations. We present algorithms that consume and analyze API features, to automatically determine whether two given APIs are syntactically compatible, and if so, how difficult it is to port an application among them. We evaluate a prototype of this
approach using randomly generated APIs to measure the sensitivity to the parameters we employ, and using competitive, publicly available APIs to determine its efficacy on real-world APIs. Finally, we show that computation of our porting effort metric introduces minimal overhead, making it sufficiently practical to include in a developer’s tool chain.

This work was funded in part by NSF (0751315, 0905237, and 1218808) and NIH (1R01EB014877-01).

10. References


Authors
Hiranya Jayathilaka is a PhD student in the Computer Science Department at UC Santa Barbara (UCSB). His research interests include distributed systems and web/cloud services. Hiranya received a BS degree in Engineering from the Univ. of Moratuwa, Sri Lanka.

Alexander Pucher is a PhD student in Computer Science at UCSB, having received his MS degree from TU Vienna, Austria. His research focuses on highly resource-efficient cloud infrastructures and cross-platform interoperability.

http://www.hipore.com/jsc/
Dr. Chandra Krintz is a Professor in the Computer Science Department at UCSB. Her research interests include cloud platforms and programming systems, and she is the progenitor of the open source cloud platform-as-a-service, AppScale. She holds MS and PhD degrees from UC San Diego.

Dr. Rich Wolski is a Professor in the Computer Science Department at UCSB. His research interests include cloud infrastructures and scientific computing, and he is the progenitor of the open source cloud infrastructure-as-a-service (IaaS), Eucalyptus. He holds MS and PhD degrees from UC Davis.
COST-EFFECTIVE SERVICE NETWORK PLANNING FOR MASS CUSTOMIZATION OF SERVICES

Zhongjie Wang, Nan Jing, Fei Xu, Xiaofei Xu
School of Computer Science and Technology, Harbin Institute of Technology, Harbin, China
rainy@hit.edu.cn, jingnanjib@sina.com, xf_01001@126.com, xiaofei@hit.edu.cn

Abstract

In real-world scenarios, massive demands might be simultaneously raised by massive customers. Mass customization (MC) of services is a cost-effective approach to deal with this scenario by offering a customizable composite service that satisfies as many personalized requirements as possible. In this paper, Service Network Planning (SNP) is employed to represent a customizable composite service, so that MC of services is transformed into a Service Network Planning (SNP) problem. A heuristic algorithm called Iterative Enhancement based Algorithm (IEA) is proposed for SNP. Massive requirements are sorted in terms of their potential benefit in the descending order, and for each requirement to be dealt with, the algorithm iteratively enhances the preceding constructed SN by the least cost instead of constructing a new one, so that the final constructed SN keeps the minimal size and complexity and reaches cost-effectiveness. For comparison, two traditional service composition based algorithms called Solution Consolidation Algorithm (SCA) and Requirement Grouping based Algorithm (RGA) are presented. Experiments are conducted to prove the superiority of the IEA, and factors that impact the performance of IEA are exploited elaborately.

Keywords: Service Composition, Mass Customization of Services, Service Network; Iterative Enhancement Algorithm, Cost-Effectiveness

1. INTRODUCTION

Mass customization (MC) refers to the design and production of personalized or custom-tailored goods or services to meet customers' diverse and changing needs, with the objective that every customer can have exactly what he wants by adding and/or changing certain functionalities of a core product (BusinessDictionary, 2014). It is of great significance to keep a wide variation in service offerings so that the personalized demands of different customers are met at high levels and at lower prices.

If we regard web services as the basic parts and a composite service constituted by multiple web services as a customizable product, it is apparent that MC of services has a similar philosophy as the MC of products, i.e., to assemble a set of fine-grained components into a coarse-grained composite solution. Nevertheless, the differences are obvious, too: (1) the reusable parts of customized products are manually designed by product designers, while available web services come from a mass of independent service providers who publicize their services on the Internet. This indicates that the MC of services is usually conducted in an open environment and there is no pre-defined reference architecture. (2) The quantity of components being a part of a customizable product is usually limited, while available web services on the Internet are comparatively numerous and their potential compositions are almost infinite. This implies that the customization degree of services is much higher than the one of the products and, as a side effect, the customization difficulty and cost is rather higher, too.

In a word, to realize the mass customization of services in the open Internet environment in a cost-effective and efficient way, is a challenging problem in services computing research.

Research of this paper aims at a typical broker-based service mass customization scenario. Numerous services are accumulated on an open platform of a broker, and they might either be publicized by their providers or be proactively collected by the broker. Customers raise their requirements to the platform, and the broker selects proper services and composes them into composite services to satisfy these requirements.

A widely adopted tactic in practice is "service product", i.e., the broker offers a composite service that owns particular customizable features, and each customer customizes these features according to his own preferences. For the broker, the cost of preparing such products is comparatively low; by contrast, on account that these customizable features are manually designed by the broker, the customization degree is limited.

Another tactic is "service composition" which goes to the other extreme, i.e., complete personalization. A composite service is not prepared in advance, but after a specific requirement is raised. Compared with service product, its personalization degree is higher, but its cost is also higher because each composite service fits perfectly for the functionality, QoS and context constraints of one specific requirement, therefore multiple composite services are needed for multiple requirements. Another deficiency is the efficiency: online service selection and composition from a large number of candidate web services is a time-consuming work.

To sum up, there is a tradeoff between the personalization/customization degree and the cost-effectiveness of a mass customizable service.
Service Network (SN) is a good choice of facilitating the MC of services by combining the two tactics together to exert their respective superiorities. Researchers have found there are explicit and implicit correlations among existing web services; as a consequence, these web services can be interconnected to form an SN which possesses many more customizable features (e.g., multi-source parameters and compound service nodes, which are introduced in Section 3.2) than "service product". Besides, due to the similarities between multiple personalized requirements, there are some frequently occurring composition patterns in the corresponding composite services, and an SN also includes such patterns that could be shared and reused by different requirements. For each incoming requirement, the SN is customized and a concrete composite service is identified.

This paper focuses on the Service Network Planning (SNP) problem, i.e., given a set of customer requirements and a set of available web services, an SN is built to meet these requirements in a mass customization way with the highest profit and the lowest cost. There are two objectives to be pursued in this problem:

1. The degree to which massive personalized requirements could be satisfied by the customization of the SN. From the perspective of sociology, the crowd behaviors usually exhibit the "continuity" along with the time, i.e., there are common characteristics among the requirements appearing in consecutive time. This indicates that, if the SN could well satisfy the recent emerging requirements, then the following-up ones might be met well with a great probability. The higher the personalization degree to which the SN satisfies the requirements is and the more is the number of requirements that the SN could satisfy, accordingly the higher is the customer satisfaction degree, and as a consequence, the higher is the direct and potential benefit the SN brings to the broker.

2. Cost-effectiveness of the SN. The cost of an SN includes the Planning Cost (PC), Customization Cost (CC) and Usage Cost (UC). PC is the cost at which the broker investigates the reputation of the services and negotiates with the service providers whose services are included in the SN, and the cost of connecting the selected service together as a network. CC refers to the cost of customizing the SN to satisfy each requirement. UC refers to the price of the final customized composite services offered to the customers. All the three cost are closely related to the size and complexity of the SN, the economic attributes of the services in the SN, and the number of customizable features in the SN.

There are three challenges in SNP problem:

1. Under the situation that the number of available web services is very large, even the traditional one-requirement-oriented service composition problem is proving to be NP-hard, moreover the SNP problem in which n requirements are considered in the meanwhile. How to design a highly customizable and cost-effective SN in an acceptable time complexity?

2. The high personalization degree requires that the SN is large enough to include more customizable features, while high cost-effectiveness is diametrically opposed. How to look for the compromise between the two objectives so that both are relatively acceptable?

3. A personalized requirement includes two types of constraints, i.e., functionality and QoS. In most of the conventional service composition approaches, either the functional or QoS requirements are to be individually handled. How to synthesize both together?

For (1), service composition research has made effective explorations. For example, the skyline-based approach filters out infeasible services before composition so that the search space is significantly reduced; the online-and-offline combined approach makes full use of the historical composition records to find out common composition patterns to narrow the search scope; and the approximate approaches, which have received the most attention, look for an acceptable solution in a limited time. We will use climbing-hilling combined with branch-and-bound to reduce the time complexity caused by the large number of candidate services.

For (2), we adopt an iterative enhancement strategy to look for an optimal tradeoff between the customization degree and cost-effectiveness. Our approach does not deal with the requirements one by one, but leverages the similarities and correlations among requirements so that the SN is iteratively constructed and reinforced using heuristic strategies to ensure that the least amount of services are composed to achieve the highest mass customization competency.

For (3), in our previous work we have proposed QoS-oriented and functionality-oriented SN planning, respectively. In this paper we will combine them together.

Briefly, the contributions of this paper are as follows: (1) we elaborate the service mass customization scenario which aims at the satisfaction of massive personalized requirements (functionality and QoS) and combine two traditional mass customization strategies together; (2) we propose a Service Network based approach to address the service mass customization issue, where an SN is a customizable composite service that could satisfy multiple personalized requirements; (3) we put forward a Service Network Planning algorithm in which the iterative enhancement strategy is leveraged to construct an SN and achieve an optimal tradeoff between the customization degree and cost-effectiveness.

The rest of this paper is organized as follows. Section 2 surveys related work; section 3 defines the logical model of SN and clarifies how it facilitates mass customization; section 4 gives the mathematical model of the SNP problem; section 5 puts forward three potential approaches for SNP problem and the corresponding algorithms; section 6 is the experiments and comparative analysis; and finally is the conclusion and future work.

http://www.hipore.com/ijsc
2. Related Work

2.1 One-Requirement-Oriented Service Composition

There are two types of service composition (SC) problems: the one with end-to-end QoS constraints (called QoS-aware service composition, QSC), and the one with semantics/functionality constraints (SSC).

QSC is a multidimensional, multi-objective, multi-choice knapsack problem (M3MKP) to look for the optimality of a single or multiple objectives under the multi-dimensional QoS constraints. Global optimality can be found by methods such as integer programming, but the most commonly used approach tries to look for acceptable solutions in the relatively lower time complexity using local search heuristics (Luo et al., 2011), or to make use of the historical composition records to find the common composition patterns (Wang, Wang, & Xu, 2012). Representative QSC methods include: greedy-based local search method (Zeng et al., 2004), skyline-based method (Alrifai, Skoutas, & Risse, 2010), genetic algorithm (Canfora et al., 2005), global and local search combined method (Alrifai, Risse, & Nejdl, 2012), multi-constraint optimal path based method (Yu, Zhang, & Lin, 2007), etc.

SSC is a typical AI planning problem. Given an initial and a target state, the planning methods identify a composite path for the transition from the initial to the target. Compared with QSC, in SSC there is not a pre-existing service process. But the difficulty is the same, i.e., when the number of candidate services is large, the search space is so huge that there are no effective algorithms with polynomial complexity (Lin et al., 2012). Various AI planners have been adopted for this challenge (Hatzis et al., 2013). Rao, & Su (2005), Peer (2005) and Oh, Lee, & Kurmara (2006) have made comprehensive surveys on SSC research. Representative methods include: GraphPlan-based method (Zheng & Yan, 2008), heuristic rule based search (Hoffmann, 2000), and dynamic planning approaches (Kuzu & Cicelki, 2012).

In spite that QSC and SSC have achieved a great in theory, they are insufficient to be directly applied to the mass customization scenario, i.e., to deal with massive personalized requirements simultaneously. For example, the time complexity is more unacceptable because multiple requirements have to be dealt with one by one.

2.2 Multi-Requirement-Oriented Service Composition

A few research work have been conducted for the multi-requirement-oriented service composition problem (MSC), with the philosophy of transforming MSC problems into multiple traditional QSC/SSC ones by requirement grouping or QoS constraint decomposition, and meanwhile decreasing the execution times of QSC/SSC algorithms.

For example, Cardellini et al. (2007) proposed a method to group the multiple arrival requirements into a set of “flows” according to the similarities of QoS constraints, and requirements in the same flow are fulfilled by one composite service; Wang, Xu, & Xu (2012) presented an algorithm named Multi-Compositions-for-Multi-Requirements (MC4MR) to construct m (≤n) composite services for n requirements with different QoS constraints to ensure the optimal cost-effectiveness. Jin et al. (2012) and Zou et al. (2013) found that those candidate services having higher cost performance will be more frequently selected to satisfy multiple requirements, and as a consequence, these services might be with higher load which would possibly violate their physical constraints. They proposed the methods to decompose the global QoS constraints into a set of local QoS constraints on each task and again decompose in terms of different users, then finally transform into the traditional QSC problem. But for n requirements, the same number of composite services must be constructed. Similar work could be found in (Liu et al., 2011), where a similar problem called global optimal service selection for multiple requesters (GOSSMR) is presented.

2.3 Methodology and Framework for Service Mass Customization

Research on the mass customization of services (Kannan & Healey, 2011; Hu et al., 2013) are generally conducted from the methodology and framework of service engineering, exploring the mechanism of how to develop a customizable service system for massive personalized requirements, instead of the narrow-sense composite services in service composition problem. Further, the customizable features are much broader, for example, in SaaS, a customizable feature might be located in the UI layer, business logic layer and data layer, and each customer makes configuration by his own preference to get a highly personalized SaaS instance. Typical techniques to implement such customization are meta-models and policy-based models (Shim, 2011).

This philosophy originates from the domain engineering and software product line (SPL) practice in software engineering (Alferez & Pelechano, 2011; Mohabbati et al., 2013). Two representative methods of this kind are configurable software engineering methodology (Becker et al., 2009) and Service Family (Moon et al., 2010; Seung et al., 2011). Semantic ontology (Liang et al., 2011), VxBPEL (Sun et al., 2010), and feature (tree) model (Nguyen & Colman, 2010), etc, are invented to describe the variable/customizable points and the dependencies between them (Hadayullah, Koskimies, & Systa, 2009; Nguyen, Colman, & Han, 2011; Weidmann et al., 2011). These variations are then mapped to the implementation in a model-driven approach (Bucchiarone et al., 2010) and infert the service system with self-adaption capacities by techniques such as autonomic agent and autonomic event triggering.

http://www.hipore.com/ijsc
Nevertheless, the service systems developed by the above-mentioned methods are essentially "closed", meaning that those customizable features are mostly identified and designed by the service designers based on their own experiences and domain knowledge. This limits the customization competency of service systems and violates the "open" characteristic of services.

2.4 Service Network Approaches
In order to improve the inadequate customization competency, a new approach called Service Network (SN) is emerging in recent years. Compared with the previous methods, it is not a single organization who dominates the design of the customizable service systems, but it is based on the open Internet environment and tries to build an open service network using an "exhaustive spreading" mechanism to aggregate various isolated web services and open APIs into an inter-connected network based on the similarities and correlations among the services. That is, an SN comes into being not so much by "elaborate design" as by "naturally growing".

Researchers used several different names for this phenomenon, such as Service Network (Chen, Han, & Feng, 2012), Service Eco-Systems (Huang, Fan, & Tan, 2012), Composition Service Network (Tao et al., 2012), Open Semantics Service Network (OSSN) (Cardoso, Pedrinaci, & De Leenheer, 2013), Global Social Service Network (Chen, Paik, & Huang, 2014), etc. A consensus is that, the more intensive business interoperability promotes the dynamic and complex interconnections between web-based software services, and a service-centric web is emerging (Danylevych, Karastoyanova, & Leymann, 2010). It has been found that an SN is a scale-free and small-world network composed of a few active services and a great amount of silent services, indicating that it follows the power law (Tao et al., 2012; Hwang, Altmann, & Kim, 2009; Kil et al., 2009). In essence, an SN represents the crowd intelligence when massive users spontaneously use these services.

Liang, Chen, & Feng (2013) adopted a snowball-based automatic service composition by exploring the semantics relations among massive web services, and based on the tracing of these potential relations, new services are constantly added into current SN so that it grows. Jung (2011) proposes a service chain identification method to discover explicit and implicit relations between services, and then uses Social Network Analysis (SNA) approach to merge multiple service chains for the flexible satisfaction of business demands. Maamar, Hacid, & Huhns (2011) and Chen, Paik, & Huang (2014) give web services the "sociability" competency, i.e. each web service knows which other web services collaborate, substitute, or compete with itself, so that the social networking approaches are utilized to build the SN. Common integration patterns are to be found in the service eco-systems to improve an SN's performance (Han, Chen, & Feng, 2014).

It is assumed that service network is a promising approach for mass customization of services. In the future, service network will become a social infrastructure, and when a service provider publicized a new service, it will be voluntarily connected into the SN; and along with more users' involvement into the SN, some relations between services might be strengthened and others be weakened. Let us envision the time when service network is like the Internet (connecting massive servers around the world) and social network (connecting massive people around the world); it connects all the publicized services around the world, no matter who offer the services and who use them. In this way, the high cost led by the one-requirement-oriented service composition is avoided. This is the reason why we use service network to facilitate the MC of services.

3. SERVICE NETWORK

3.1 Logical Architecture and Definition
Figure 1 shows an example of a simplified service network. From logical structure's point of view, a SN is composed of four parts: Input, Services, Output, and directed connections between them.

Formally, a service network is defined by \( SN=\{I, O, S, F, G, H\} \), where:

\( I=\{p_i\} \) and \( O=\{p_j\} \) are the input and output parameters of SN, respectively. Each parameter represents a distinct data item offered or expected by users, and is described in the form of standard public ontology.

\( S=\{s_i\} \) is a set of abstract service nodes, and each \( s_i=\{l_i, O, A_i\} \) is composed of the input parameters \( l_i \), output parameters \( O_i \) and one or multiple concrete web services \( A_i \). \( |A_i|=1 \) implies there is only one candidate service for \( s_i \), and \( |A_i|>1 \) indicates \( s_i \) has multiple functionally-equivalent candidates which have the same input and output but different QoS. The \( j \)-th concrete service is represented by \( a_{ij}=\{l_i, O_j, T_{ij}, R_{ij}, P_{ij}\} \) where \( T_{ij}, R_{ij}, P_{ij} \) are the values of \( a_{ij} \)’s execution time, reliability and price, respectively.
\[ F = \{ f_i \} \] is a set of directed edges pointing from the input parameters to the service nodes, each edge \( f_i = p_i \rightarrow s_i \) (\( p_i \in I \), \( s_i \in S \)) indicating an input parameter \( p_i \) is transferred to the service node \( s_i \). \( G = \{ g_{ij} \} \) is the directed edges from service nodes to the output parameters, each \( g_{ij} = s_i \rightarrow p_j \) (\( s_i \in S \), \( p_j \in O \)) indicating the service node \( s_i \) will transfer its output parameter \( p_j \) directly to the output of \( SN \). \( H = \{ h_{ij} \} \), and \( h_{ij} = s_i \rightarrow s_j \) (\( s_i, s_j \in S \)) indicates that a parameter \( p_k \) is transferred between two service nodes \( s_i \) and \( s_j \). The three types of edges jointly transfer the input \( I \) to the output \( O \) layer by layer.

To note that \( O(SN) = \bigcup_{i \in S} O(S_i) \). This implies that any output parameters of any service nodes in the \( SN \) could be as the output of \( SN \).

An \( SN \) tends to come into being either spontaneously or planned by a predominant service provider on its own initiative. All the concrete services included in the service nodes are the open publicized web services or APIs. Two inter-connected service nodes have the "related-to" relation, and two concrete services belonging to the same service nodes have the "similar-as" relation.

### 3.2 How SN Enables Mass Customization

An \( SN \)'s customization refers that, given a requirement with functionality and QoS constraints, we look for an \( SN \)'s sub-network that satisfies both constraints. Here we briefly introduce why the \( SN \) is customizable.

\[ \forall s_j \in S, \exists p_k \in I(s_j), \quad \Phi(s_j, p_k) = \{ f_{ij} \mid f_{ij} = p_i \rightarrow s_j \} \cup \{ h_{ij} \mid h_{ij} = s_i \rightarrow s_j \} \]

is the set of directed edges from the input of \( SN \) or other service nodes to \( s_j \) and that transfer the parameter \( p_k \). This implies that all the edges in \( \Phi(s_j, p_k) \) have the "or" relations, i.e., when \( SN \) is customized, as long as at least one edge in \( \Phi(s_j, p_k) \) is kept in the final sub-network, will the input parameter \( p_k \) of \( s_j \) be provided. For example in Figure 1, the service node \( s_3 \) has three incoming edges all transferring the parameter \( p_3 \), one from the input of \( SN \), the second from \( s_1 \), and the last from \( s_2 \). Similarly, \( \forall p_k \in O \), all the directed edges in \( \Theta(p_k) = \{ g_{ij} \mid g_{ij} = s_i \rightarrow p_j \} \) have the "or" relations, such as the output parameters \( p_9 \) and \( p_{10} \) in Figure 1. The existence of the relation "or" implies that \( SN \) has multiple ways of producing expected data requested in a specific requirement.

\textbf{Def. 1 (Multi-Source Parameter)} \( \forall s_j \in S, \exists p_k \in I(s_j), \) if \( |\Phi(s_j, p_k)| \geq 1 \), then \( p_k \) is a multi-source parameter. \( \forall p_k \in O, \) if \( |\Theta(p_k)| \geq 1 \), then \( p_k \) is a multi-source parameter, too.

Secondly, a compound node is also customizable, i.e., a concrete service is to be selected from \( A \) to satisfy the QoS constraint in a specific requirement. In Figure 1, the compound service node is shown as double-line cycles to be distinguished from those non-compound service nodes where |\( A \)|=1.

\textbf{Def. 2 (Compound Service Node)} \( \forall s_j \in S, \) if |\( A \)|>1, then \( s_j \) is a compound service node.

To sum up, there are two mechanisms that \( SN \) holds to facilitate the mass customization: (1) the variety how a specific parameter is produced, i.e., the multi-source parameter. (2) the variety that a specific service node exhibits different QoS levels, i.e., the compound service node. The former makes the customization results have personalized process structures, while the latter makes the results exhibit personalized QoS levels. From this point of view, \( SN \) can be considered as the fusion of two traditional service composition approaches (SSC and QSC): AI planning based SSC methods (e.g., GraphPlan) are good at producing the variety of service inter-connections (i.e., multi-source parameters), and QSC methods are good at producing the variety of global QoS of the final solutions (i.e., to select proper concrete services for the composition), respectively.

### 4. Problem Definition

#### 4.1 Personalized Requirement

A personalized requirement is composed of three parts: functionality constraints, QoS constraints, and Willing to Pay (WTP). It is denoted by \( r = F^p, O^q, W^r \), where:

\[ F^p = \{ p_i \} \] and \( O^q = \{ p_j \} \) are the data set offered and expected by a customer, respectively. They jointly represent the functionality constraints: the customer knows his initial state (\( F^p \)) and expected state (\( O^q \)), but does not know how to achieve the transformation from \( F^p \) to \( O^q \).

\[ Q^r = \{ \langle QParam, QCons, > \} \] is a set of QoS attributes and the corresponding constraints acting on the QoS of the final customized solution. For example, \( \langle \text{Time}, \leq 10 \text{hours} \rangle \) and \( \langle \text{Reliability}, \geq 0.98 \rangle \) are two constraints on the execution time and reliability, respectively.

\[ W^r \] is the amount of money the customer is willing to pay if all the expected data in \( O^q \) could be generated and all the constraints in \( Q^r \) could be satisfied by the final customized solution.

#### 4.2 Candidate Services

A candidate service is abstractly defined as \( a = F, F^p, O^q, Q^r, NC, UC, > \), where \( F \) is the functional description of \( a \) in the form of standard ontology. \( F^p \) and \( O^q \) are \( a \)'s input and output parameters, respectively, and \( Q^r = \{ \langle QParam, QValue, > \} \) is a set of QoS attributes and the corresponding values declared by \( a \)'s provider.

\( NC \) and \( UC \) are two cost-related metrics. \( NC \) is the Negotiation Cost, referring to the cost that a broker pays for investigating the functionality, QoS, reputation, etc. of \( a \), and for negotiating with \( a \)'s owner (service provider) to get the permission that \( a \) is to be included in the \( SN \). \( UC \) is the Usage Cost, the price that a customer pays to the provider if \( a \) participates in the customized solution that satisfies his requirement. Some web services and APIs are free, i.e.,

http://www.hipore.com/ijsc
UC(a)=0. For those paid services, UC(a) is the publicized price of a. No matter whether a is a free or paid service, there always has NC(a)>0. In the past service computing research, people usually focus on UC but basically ignore NC. To note that, NC(a) is paid once for the first time when a is added into SN and never needs to be recalculated regardless of how many times it is used. But every time a participates into the satisfaction of a requirement, UC(a) is paid once, i.e., pay-per-use.

For two candidate services \( a_i \) and \( a_j \), \( F(a_i)=F(a_j) \) indicates both offer the same functionality, denoted by \( a_i \Leftrightarrow a_j \). On the premise of \( a_i \Leftrightarrow a_j \), if \( I^r(a_i)=I^r(a_j) \) and \( O^r(a_i)=O^r(a_j) \), then \( a_i, a_j \) have the same input and output, which is denoted by \( a_i \Leftrightarrow a_j \). \( \forall s_i \in SN \), all the concrete services in its \( A_i \) satisfies the relation \( \Leftrightarrow \), but have difference QoS values. If \( a_i \Leftrightarrow a_j \) holds but \( a_i \Leftrightarrow a_j \) does not, \( a_i \) and \( a_j \) cannot belong to the same service node in SN.

4.3 Problem Definition

The problem of Service Network Planning (SNP) is defined as follows. Given a set of personalized requirements \( R=\{r_i\} \) (\( k=1,2, ..., n \)) and a set of candidate services \( CS=\{s_i\} \) (\( l=1,2, ..., m \)), we build a customizable service network SN. The expected output is refined into two aspects:

1. \( SN, \) having \( I^r(SN) \equiv \bigcup_{r \in R} I^r(r) \), \( O^r(SN) \equiv \bigcup_{r \in R} O^r(r) \);
2. \( CR \subseteq R \) being the requirements that could be satisfied by \( SN. \) \( CR \subseteq R \) implies all the requirements are satisfied, \( \forall r_i \in CR \), \( \exists C_i \) is a sub-network of \( SN \) and satisfies the functionality and QoS constraints in \( r_i \). From the service composition's standpoint, each \( C_i \) is a composite service.

This is a combinatorial optimization problem. Because there might be massive potential \( SN \) composed by the massive services in \( CS \), the search space is large. The objective is to build the \( SN \) that satisfies as many requirements in \( R \) as possible with minimal cost. In other words, to maximize the benefit that the broker yields from the \( SN. \) This is called ”cost-effectiveness” and denoted as:

\[
\text{max Profit}(SN, CR) = \text{Benefit}(CR) - \text{Price}(SN, CR) - \text{CC}(SN)
\]

\( \text{Profit}(SN, CR) \) is the net earnings from the construction of \( SN \) for satisfying the requirements in \( CR. \) Shown in Figure 2, it has three constituent parts:

- \( \text{Benefit}(CR) = \sum_{r \in R} W_r \) is the total revenue received from the customers whose requirements are included in \( CR; \)
- \( \text{Price}(SN, CR) = \sum_{a \in \cup_{s \in SN} A} (UC(a) \times |CR(a)|) \) is the Usage Cost (UC) that the broker pays to the service providers whose services participates into the satisfaction of the requirements in \( CR, \) and \( CR(a_j) = \{ r_j \ | \ r_j \in R, a_j \in CN_r \} \) is the set of requirements where \( a_j \) participates.
- \( \text{CC}(SN) = \sum_{r \in R} NC\{a_j\} + \mu \times H(SN) \) is the planning and maintenance cost borne by the broker. The former part is the Negotiation Cost (NC) for the services included in \( SN, \) and the latter is the maintenance cost for the inter-connections between service nodes in \( SN, \) where \( \mu \) is the unit cost of service connections.

Constraints: \( \forall r_i \in R, \forall QParam_j \in Q^r, \) the value of \( CN_i \)'s global \( QParam_j \) satisfies \( QCons_j. \)

From the optimization objective three conclusions can be drawn:

- The larger is the number of requirements that can be satisfied (i.e., |CR|) and the larger is the \( W^R \) that the satisfied requirements offer, the larger Profit(SN, CR) might be. Therefore, the SNP algorithm should give higher priority to those requirements that could bring more benefit;
- The smaller is the number of services included in SN and the smaller is the number of inter-connections between these services, the larger Profit(SN, CR) might be. Therefore, the SNP algorithm should cautiously import services and interconnections into SN unless they result in the profit growth of the SN;
- The lower UC and NC do the services in SN have and the higher is the number of requirements that each service participates in, the larger Profit(SN, CR) might be. Therefore, the SNP algorithm should try to use those services with higher cost performance and higher reusability.

But when we get down to the details, above three principles might contradict each other. For example, a requirement with higher WTP usually has more strict functionality and QoS constraints, and this implies the selected services should have better quality but consequently, higher price. Another example is: in order to use less services to satisfy more requirements, those services with higher quality will be usually selected to cover more requirements, but for those requirements with more relaxed QoS constraints, such service selection is a waste and possibly leads to the reduction of the profit; however, if we follow the "just-enough" principle to select the most suitable

http://www.hipore.com/ijsc
services to exactly satisfy the QoS constraints of each requirement, the number of services and inter-connections will inevitably high. The challenge is to make an acceptable compromise on these conflicting objectives and plan an optimal SN.

5. Optimization Algorithms

5.1 Three Potential Approaches

Compared with traditional service composition approaches where the number of requirements and the number of final composite services is 1:1, the SNP problem is that multiple requirements share one SN (i.e., n:1). A critical step is how to transform n:1 SNP into one or several 1:1 service composition problem, then to apply traditional service composition algorithms to solve them, respectively. Here we use three strategies for such transformation:

- **Strategy 1**: To construct a composite service for each requirement, and then merge these composite services into an SN;
- **Strategy 2**: To group requirements into m (1≤m≤n) virtual ones, and then use Strategy 1 to plan the SN;
- **Strategy 3**: To sequentially deal with n requirements one by one and always keep one SN by iteratively enhancement.

Processes of the three strategies are shown in Figure 3(a)(b)(c), in which ORSC refers to the traditional One-Requirement-oriented Service Composition algorithm.

5.2 Solution Consolidation based Algorithm

This strategy is the direct application of traditional 1:1 service composition approaches into the MC scenario, i.e., using ORSC to plan a solution for each requirement, then merging all the solutions together and eliminating the repeated and unnecessary service nodes. The algorithm is called Solution-Consolidation-based Algorithm (SCA) and is listed below.

Algorithm 1: Solution-Consolidation-based Algorithm (SCA)

<table>
<thead>
<tr>
<th>Input:</th>
<th>Output: SN, CR, CNi (1≤i≤n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) for ∀ri∈R</td>
<td>(2) CNi ← ORSC (ri, CS)</td>
</tr>
<tr>
<td>(3) if CNi≠∅</td>
<td>(4) CR ← CR∪{ri}</td>
</tr>
<tr>
<td>(5) end for</td>
<td>(6) SN ← Merge (CN1, CN2, ..., CNn)</td>
</tr>
</tbody>
</table>
| (7) return SN, CR, CNi

The advantage of SCA is straightforward: it is simple and a traditional ORSC algorithm is applied n times with no need for designing a new algorithm. The disadvantages are obvious, too, i.e., it does not consider the correlations between requirements (indicating that the selected services in step (2) are only used to satisfy one requirement) so that the final consolidated solution in step (6) will include a large number of services with high complexity of interconnections and at high cost.

The time complexity of SCA is subject to the number of requirements (n), being equivalent to n times of the one of ORSC algorithm.

5.3 Requirement Grouping based Algorithm

This algorithm is based on SCA, but further considers the correlations and similarities between the n requirements and divides them into groups according to the coverage relations. A virtual requirements is constructed as the representative of all the requirements belonging to the same group, and ORSC is applied for each virtual requirement. In this way, the number of requirements is compressed and the time complexity is reduced. The philosophy is straightforward: if a composite service can meet a requirement with strict constraints, then it can surely satisfy other requirements with more relaxed constraints. Here we firstly define two coverage relations between requirements and use them for requirement grouping.

**Def. 3** *(Functionality Coverage between two requirements)*. For two requirements r_i and r_j, the conditions \( F_r(\{r_i\}) \subseteq F_r(\{r_j\}) \) and \( \delta_r(\{r_i\}) \supseteq \delta_r(\{r_j\}) \) implies that a composite service that could satisfy the function constraint of \( r_i \) must satisfy the one of \( r_j \). We call that \( r_i \) "functionally covers" \( r_j \), denoted as FCover(\( r_i, r_j \)).
Def. 4 (QoS Coverage between two requirements). For \( r_i \) and \( r_j \) where there is \( FCover(r_i, r_j) \), if each QoS constraint in \( r_j \) is more strict than the constraint on the same QoS attribute in \( r_j \), we called that \( r_i \) "QoS covers" \( r_j \), denoted as \( QCover(r_i, r_j) \).

In terms of the requirement grouping, there are two steps:

1. Partition \( R \) into multiple mutually disjoint subsets \( \{ R_1, R_2, \ldots, R_p \} \) by the \( FCover \) relation, i.e., for each \( \forall r_i \in R_k \) \((1 \leq k \leq p)\), there is at least one \( r_i' \in R_k \) that makes \( FCover(r_i', r_i) \), and for \( \forall r_i \in R_k \), both \( FCover(r_i, r_j) \) and \( FCover(r_j, r_i) \) are false; (2) Similarly, for each \( k \in [1, p] \), partition \( R_k \) into mutually disjoint subsets by the \( QCover \) relation.

Finally \( R \) is grouped into \( \{ R_1, R_2, \ldots, R_m \} \) and \( 1 \leq p \leq m \leq n \).

Algorithm 2: Requirement-Grouping-based Algorithm (RGA)

**Input:** \( CS, R \)

**Output:** \( SN, CR, CN_i(1 \leq i \leq n) \)

1. Partition \( R \) into \( \{ R_1, R_2, \ldots, R_m \} \) w.r.t. \( FCover \) and \( QCover \).

\[ \bigcup_{i=1}^{m} R_i = R \]

2. for \( \forall i \in [1, m] \)

3. \[ v_{r_i} \leftarrow \left( I^s_i, O^s_i, Q^s_i, W^s_i \right) \]

\[ \text{where } I^s_i \leftarrow \bigcap_{r_i \in R_i} I^s_r, \quad O^s_i \leftarrow \bigcup_{r_i \in R_i} O^s_r, \quad W^s_i \leftarrow \sum_{r_i \in R_i} W^s_r, \quad Q^s_i \leftarrow \text{Strictest } Q^s_{r_i} \]

4. end for

5. \( VR \leftarrow \{ v_{r_1}, v_{r_2}, \ldots, v_{r_m} \} \)

6. return \( SCA(CS, VR) \)

Step (1) partitions the initial requirements \( R \) into \( m \) subsets and each subset is a partially ordered set in terms of \( FCover \) and \( QCover \). For each \( R_j \), step (2) and (3) construct a virtual requirement \( v_{r_j} \) with the least input parameters, the most output parameters, the total WTP, and the strictest QoS constraints of all the requirements in \( R_j \). Step (6) invoke \( SCA \) algorithm to plan the final \( SN \).

Compared with \( SCA \), \( RGA \) compresses the requirements and transforms the initial \( n:1 \) composition problem into an \( m:1 \) (\( m \leq n \)) one. Therefore, the time complexity decreases \( n/m \) times. In the worst case where \( m = n \), i.e., there are not any \( FCover \) and \( QCover \) relations among all the requirements, so \( SCA \) and \( RGA \) have the same efficiency. In the best case where \( m = 1 \), i.e., all the requirements are mutually covered each other, the time complexity of \( RGA \) is equivalent to the one of \( ORSC \). Besides, the number of service nodes and connections in the final \( SN \) has lowered a great deal because multiple requirements in the same group share the common composite solution.

But as mentioned before, for two requirements \( r_i, r_j \) in the same equivalence class, if \( r_j \)'s QoS constraints is extraordinarily stricter than \( r_i \)'s, the virtual requirement must follow the constraints of \( r_i \). This violates the "just-enough" principle and results in cost-ineffectiveness. From this perspective, the output \( SN \) is not optimal. On the other hand, \( RGA \) only considers \( FCover \) and \( QCover \) relations which look somewhat strict. For example, suppose \( r_i \) expects to get the output \( \{ a, b, c \} \) by offering the input \( \{ y, z \} \), and \( r_j \) expects to get \( \{ b, d \} \) by offering \( \{ x, z \} \); in \( RGA \), \( r_i \) and \( r_j \) do not satisfy the \( FCover \) relation, hence the \( ORSC \) algorithm is to be invoked twice to construct the solutions for each, respectively. However, there are indeed some similarities, e.g., both provide \( b \) and expect \( z \). To further improve the optimality, we go to the third strategy, Iterative Enhancement based Algorithm (IEA).

5.4 Iterative Enhancement based Algorithm

From above analysis, we get a heuristic rule: it had better utilize more shared services and connections that have the capacity of satisfying multiple requirements, so that the negotiation and maintenance cost are to be averaged, and consequently, the total cost is to be reduced.

Here we introduce the iterative enhancement strategy based on above rule. Suppose that a service network \( SN^{(0)} \) has been constructed for the first \( i \) requirements \( \{ r_1, r_2, \ldots, r_i \} \), we first check whether \( r_{i+1} \) could be fully satisfied by \( SN^{(0)} \). If yes, it is not necessary to invoke the \( ORSC \) algorithm for \( r_{i+1} \); otherwise, \( ORSC \) is called to import new services and connections and merge them into \( SN^{(0)} \) to get a larger \( SN^{(i+1)} \). During the \( ORSC \), it is better to reuse the existing services and connections in \( SN^{(0)} \) because their CC have already been "sunk cost", and secondly, the new imported services should follow the "just-enough" principle for the satisfaction of the constraints of \( r_{i+1} \). Such iterations continue until all the requirements have been coped with and a final \( SN^{(n)} \) is obtained. In each iteration, the criterion whether a requirement is to be satisfied or not is based on the cost-effectiveness, i.e., the satisfaction of a requirement will increase the profit of \( SN^{(n)} \).

A critical point in this process is in what order the requirements are dealt with. We use a metric "Potential Benefit" to sort the requirements with the objective that the requirements which would bring more benefit to the broker stand in the front of the queue.

Algorithm 3: Iterative Enhancement based Algorithm (IEA)

**Input:** \( CS, R \)

**Output:** \( SN, CR, CN_i(1 \leq i \leq n) \)

1. Re=SortByPotentialBenefit(R)

2. if \( |R|=1 \)

3. \( SN^{(0)} \leftarrow ORSC(r_0, CS) \), \( CN_0 \leftarrow SN^{(0)} \), \( CR \leftarrow CR \cup \{ r_0 \} \)

4. return

5. else

6. \( SN^{(n-1)} \leftarrow IEA(R \setminus \{ r_n \}, CS) \)

7. \( SN^{(n-1)} \leftarrow \text{Prune}(SN^{(n-1)}, \{ \{ SN^{(n-1)} \} \} ) \)

8. if \( O_{r_n} \subseteq O(SN^{(n-1)} \land SN^{(n-1)}) \) satisfies \( Q^s_{r_n} \)

http://www.hipore.com/ijsc
(9) \( SN^{(n)} \leftarrow SN^{(n-1)}, CN_r \leftarrow SN^{(n)}, CR \leftarrow CR \cup \{ r_n \} \)

(10) return

(11) for \( \forall a_i \in \bigcup_i \{ SN^{(n-1)} \} \)
(12) \( uFlag(a_i) \leftarrow 1, NC(a_i) \leftarrow 0 \)
(13) for \( \forall h \in H \)
(14) \( uFlag(h) \leftarrow 1 \)
(15) \( SN^{(n)} \leftarrow ORSC(r_n, CS) \)
(16) \( SN^{(n)} \leftarrow Merge(SN^{(n-1)}, SN^{(n)}) \)

5.5 Estimation of Potential Benefit of a Requirement (PB)

PB\( (r_i) \) measures the potential benefit that is produced by the composite solution of \( r_i \). It is composed of two parts: (1) the benefit received from the payment of the customer who raises \( r_i \); (2) if \( r_i \)'s composite solution can satisfy other requirements \( \{ r_{i1}, r_{i2}, \ldots \} \), then it is not necessary to construct new solutions for these requirements, and the payment of the corresponding customers is considered as \( r_i \)'s benefit, too. The larger is PB\( (r_i) \), the higher priority \( r_i \) is to be dealt with during IEA, and the higher probability the follow-up iterations fall into the case (b) with.

PB\( (r_i) \) cannot be precisely measured before \( r_i \)'s composite solution is actually constructed. Here we use an estimation based on the "similarity" between two requirements, i.e., \( D^F(r_i, r_j) \) and \( D^R(r_i, r_j) \). The former measures the similarities between the input and output parameters of \( r_i \) and \( r_j \), and the latter measures the similarities between the QoS constraints of \( r_i \) and \( r_j \). Larger similarity implies that the corresponding solutions might be more similar, and the satisfaction of \( r_i \) will have a high probability of the satisfaction of \( r_j \).

For the functional similarity between \( r_i \) and \( r_j \), \( D^F(r_i, r_j) = \xi^F\left(r_i, r_j\right) \), where \( \xi^F\left(r_i, r_j\right) = \left| O^F \cap O^F\prime\right| / \left| O^F\prime\right| \) and \( \xi^F\left(r_i, r_j\right) = \left| R^F \cap R^F\prime\right| / \left| R^F\prime\right| \), and \( D^F(r_i, r_j) \in (0, 1] \).

For the QoS similarity between \( r_i \) and \( r_j \), \( D^R(r_i, r_j) = \sum_{QParam_k \in Q'^k} \xi^R\left(r_i, r_j\right) \), where \( QParam_k \) is a common QoS attribute included in \( r_i \) and \( r_j \), if \( QCons_k(r_i) \geq QCons_k(r_j) \) (indicating \( r_i \) requests a stricter \( QParam_k \) than \( r_j \)), then \( Strict_k(r_i, r_j) = 1 \). \( D^R(r_i, r_j) \) is a non-negative integer.

To sum up, the potential benefit is estimated by:

\[ PB\left(r_i\right) = W^b_i + \sum_{r_{i1}, r_{i2}, \ldots} D^F\left(r_i, r_j\right) \frac{1}{\left| \left| \left( r_i, r_j \right) \right| \right|^3} \times W^b_j \]

where \( W^b_i \) and \( W^b_j \) are the benefit gained from \( r_i \) and \( r_j \), respectively, and \( D^F\left(r_i, r_j\right) \frac{1}{\left| \left| \left( r_i, r_j \right) \right| \right|^3} \) represents how much probability there is that the satisfaction of \( r_i \) will lead to the satisfaction of \( r_j \). The reason why \( D^R\left(r_i, r_j\right) \) is placed as an exponent of \( D^F\left(r_i, r_j\right) \) is that functional similarity has greater impact on the final solution than the QoS similarity. And the greater are the \( D^F\left(r_i, r_j\right) \) and \( D^R\left(r_i, r_j\right) \), the larger \( PB\left(r_i\right) \) is.

5.6 One-Requirement Oriented Service Composition Algorithm (ORSC)

Here we introduce the ORSC algorithm used in SCA, RGA and IEA. ORSC gets the input of a requirement and a set of candidate services, and generates a composite service that satisfies the functionality and QoS constraints in the requirement. Obviously there is not a pre-existing abstract
service process, so traditional QSC algorithms are unfit for ORSC. Here we propose the ORSC algorithm based on the traditional SSC approaches.

In AI planning based SSC algorithms, the input parameters in the requirement are considered as the initial state, the expected output parameters are considered as the target state, and each candidate service is considered as the "action", then specific AI planners are employed to look for a composite service that transform from the initial to the objective state. In the planning process, the composite solution gradually comes into being, i.e., before the functionality constraints are fully satisfied, the solution is always partial. It is easy to check the satisfiability of functionality constraints (e.g., by judging whether all the expected output parameters have been produced) but it is difficult to check the satisfiability of QoS constraints because this checking should be based on a complete solution instead of a partial one. For example, different process structures lead to different calculations of getting the global execution time from the execution time of each constituent web service included in the process.

To address this issue, we present a Planning Graph based algorithm combined with branch-and-bound and hill-climbing strategies. Aiming at a requirement $r$ and the candidate services CS, the algorithm looks for a cost-effective composite solution CN.

**Algorithm 4: ORSC**

**Input:** CS, $r$

**Output:** CN

1. $PG \leftarrow$ PlanningGraph(CS, $r$)
2. if $PG \neq \emptyset$
   1. return SolutionTreeSearch(PG, $r$)
3. return $\emptyset$

In step (1) a planning graph $PG$ is generated by the classical forward chaining approach. A planning graph is a layered directed graph, denoted by $PG=<P_0, S_1, P_1, ... , S_T, P_T, M>$ where $P_0, P_1, ..., P_T$ are parameter layers, $S_1, S_2, ..., S_T$ are service layers, and $M$ is a set of parameter transferring relations (directed edges) between parameter and service layer. $P_0$ is the "initial state" and $P_T$ is the "target state" of the planning, and $PG$ contains all the possible composite solutions of $r$. For the algorithm PlanningGraph(CS, $r$) that generates $PG$, please refer to the algorithm Compose() and Expand() in Zheng & Yan (2008) and we will not repeat here. If some expected output parameters in $r$ cannot be generated by services in CS, then the algorithm returns $PG = \emptyset$. Figure 4 shows an example of planning graph with two service layers and three parameter layers.

Next, we use the hill-climbing and branch-and-bound strategies to conduct the backward search in the planning graph and look for an optimal solution. This is called SolutionTreeSearch(PG, $r$) in step (3). $PG \neq \emptyset$ implies there must be at least one feasible solution that satisfies the functional constraints of $r$, but whether the QoS constraints are satisfied should be checked during the search.

![Figure 4. A planning graph example](http://www.hipore.com/ijsc)

**Def. 5 (Solution Tree).** A solution tree $ST = <root, N, E>$ is a layered tree representing the dynamic search process on the planning graph $PG$, where $root \in N$ is the root node of $ST$, $N=[node_i]$ is a set of nodes each of which represents a partial or complete solution obtained during the search; $<node_i, node_j> \in E$ is a directed edge between nodes, node$_i$ is generated by adding a new concrete service and the corresponding edges into node$_j$, and there have node$_i$=Father(node$_j$) and node$_j$=Children(node$_i$). If $<node_i, node_j> \in E$ and $<node_j, node_k> \in E$, then node$_j$ and node$_k$ are siblings, indicating that they are two different extensions of node$_i$.

A node in $N$ is further refined as node$_i=<SList, LPList, DPLList>$, where

- $SLList=[a_k]$ is the set of concrete services included in this solution;
- $LPList$ is a set of input parameters that are required by this solution, but cannot be offered by $r$;
- $DPLList \subseteq O^P(r)$ is a subset of $r$’s output parameters that could be produced by this solution.

For the root being a special solution, there are $SLList=\emptyset$, $LPList=O^P(r)$ and $DPLList=\emptyset$, i.e., the initial solution where no any services are selected from $PG$. Each leaf-node of $ST$ represents a feasible solution of $r$, i.e., $DPLList=O^P(r)$ and $LPList=\emptyset$. Each non-leaf node is a partial solution, i.e., $DPLList \subseteq O^P(r)$. The algorithm SolutionTreeSearch(PG, $r$) is to find an optimal leaf-node with the best profit by iteratively and heuristically extending and pruning the $ST$ according to the structure of $PG$, until there are not any nodes that can be further extended and $ST$ becomes empty. The pseudo-code of SolutionTreeSearch(PG, $r$) is given below.

**Algorithm 5: SolutionTreeSearch**

**Input:** PG, $r$

**Output:** opt

1. opt$\leftarrow \emptyset$, ST$\leftarrow \emptyset$
2. ST.root$\leftarrow <\emptyset$, $O^P(r)$, $\emptyset$>
3. while ST$\neq \emptyset$
   1. $opt \leftarrow \emptyset$
   2. ST.topof Tree$\leftarrow <opt, ST.tree>$
   3. if $opt \neq \emptyset$
      1. if $opt \subseteq O^P(r)$
         1. if $ST.tree \subseteq O^P(r)$
            1. return $opt$
         2. ST.tree$\leftarrow ST.tree - opt$
      2. ST.tree$\leftarrow ST.tree - opt$
   4. return $opt$

http://www.hipore.com/ijsc
node ← HeuristicSelect(ST) p ← ParamWithLargestLayer(PG, node.LPList) AυS ← {a | a ∈ PG ∧ p ∈ O^5(a)}

(7) for ∀a ∈ AυS

(8) child ← node.LSList ∪ {a},
    node.LPList ∪ f (a)\O^5(a), node.DPList ∪ O^6(a)

(9) if EvaluateQoS(child) = false
(10) continue
(11) if child.LPList = ∅ ∧ CalProfit(child) > opt.Profit
    opt ← child
(12) if child.LPList ≠ ∅
(13) Father(child) ← node
(15) end for
(16) if Children(node) = ∅
(17) ST ← TreePruning(ST, node)
(18) end while
(19) return opt

The variable opt keeps the optimal solution during the iterations. Step (2) creates the root node of ST as the starting search point. Step (3)–(18) is a complete iteration. In step (4) a non-leaf node with the maximum profit is greedily selected from current ST for the extension in the current iteration, which is what is called the "hill-climbing" in terms of the profit of nodes. Step (5) identifies a parameter p from LPList of node such that p has the largest layer number in PG. Step 6 finds all the concrete services AυS in which each service could produce p and might be potentially extended into node to form a new solution. Every time step (8)-(14) are executed, each concrete service a in AυS is added to node and there forms a new solution child that locates in the next layer of node. Step (8) is to construct the LSList, LPList, DPList of child. Step (9) checks whether child satisfies the QoS constraints of r. If the QoS constraints are violated, child is an illegal solution and will not be added into ST; otherwise, in step (11) the profit of child is calculated. If child is a leaf-node (i.e., its LPList = ∅ and it is a complete solution) and its profit is larger than opt, then opt is substituted by child (step (12)). If child is a non-leaf node (i.e., its LPList ≠ ∅ and it is a partial solution), then child is added into ST as a child-node of node for further extension (step (13)-(14)). This implies that, if child is not superior to opt, future extension on it becomes unnecessary, and it is not added into ST, so that the search space in G is cut down. This is what is called "branch-and bound". After the execution of step (7)-(15), ST might be extended by either adding one non-leaf node under node or without any changes. "No changes" implies that all the extended solutions of node are checked to be infeasible or have been deleted, then node makes no sense and should be deleted from ST, so the algorithm goes to step (16)-(17) to prune node and its ancestors if a node satisfies the condition that it has been extended but no child nodes are kept in ST. The iterations continue, until the tree becomes empty, indicating that all the potential and feasible solutions have been explored, and opt is the final optimal solution.

To note that, in step (9) we check whether child satisfies the QoS constraints in r. As mentioned before, it is difficult to do such checking because the solutions generated during the search might be partial. Here we classify various QoS attributes into structure-unrelated (e.g., Reliability and Cost) and structure-related ones (e.g., Execution Time). The former refers that the calculation of their values is only related to the values of the constituent services but has nothing to do with the structure of the composite service, and the latter depends on both the QoS values of the constituent services and the structure of the composite service. If child is a complete solution, we check all its QoS attributes, but if it is a partial one, only the structure-unrelated QoS attributes are to be checked. If a partial solution violates some QoS constraints, it is not necessary to make further extensions on it because more extension will bring about more serious violations of the same QoS constraints. This is also what is said "branch-and bound" in this algorithm, too.

6. EXPERIMENTS AND COMPARATIVE ANALYSIS

6.1 Data Preparation
The experiments are for the three algorithms (SCA, RGA and IEA) and comparisons are made on the execution efficiency, the complexity of the generated SN, and the profit (cost-effectiveness). They are conducted on Microsoft Windows 7 (32bit), Intel® Core i3 3.10GHz processors and 2.92GB RAM. The program is implemented in Java (Eclipse 4.3+JDK1.7).

For the candidate services, we employ two web service datasets (QWS and WS-DREAM) and a self-generated dataset. 1,000 web services are extracted from the open datasets and their functionality/input/output are semantically unified. To ensure the impartiality of the results, 1,500 new services are generated by the following rules:

- 1,000 distinct data parameters are firstly selected from the parameters of the services from QWS and WS-DREAM. They are divided into 20 disjoint groups according to the similarities in their semantics;
- For each service, 1-11 parameters are randomly selected as its input, and 16-26 ones are randomly selected as its output. Either the input or output parameters of a service is selected from at most 3 different parameters groups. The number of candidate services having the same functionality and the same input/output parameters are 4 on average.
- The values of three QoS attributes (Execution Time, Reliability and Price) of a service range arbitrarily in [1,1000] seconds, [0.8,0.99], and [78,96335] dollars, respectively. For the Price (i.e., UC), its value is partially based on the values of the other two attributes;

http://www.hipore.com/ijsc

25
in other words, a service with better performance is likely to have higher Price, and vice versa.

- The value of NC of a service ranges in [800,969000] dollars. It is not randomly generated but is based on the value of the Price (i.e., the UC). The NC-to-UC ratio ranges from [0.25,10]. This is to simulate the reality: for some real-world services, it takes much cost to investigate them and negotiate with their providers, while for some others, this cost is low. Totally 1,000 requirements are prepared. As there are not off-the-shelf requirement datasets, the requirements are generated by the rules below:

  - The number of input parameters of each requirement ranges in [7,27], and the number of its output parameters is 5;
  - The QoS constraints in a requirement are in terms of the Execution Time and Reliability, the value of which range in [3100,11100] seconds and [0.1,0.52], respectively. Some requirements are endowed with more strict QoS constraints, and others are with more relaxed ones.
  - Concerning the WTP of a requirement, similar as the Price of a candidate service, there is also a metric called QoS/WTP Alignment Degree. The WTP of a majority of the generated requirements is aligned with the strictness of its QoS constraints (i.e., the more strict are the constraints, the larger is the WTP). A small number of requirements are with strict QoS constraints but smaller WTP, or with relaxed QoS constraints but larger WTP.

Based on these data, total five experiments are designed. Experiment 1 is to make comparison on the three SNP algorithms (SCA, RGA and IEA) in terms of the time complexity, the complexity of the generated SN, and the benefit/cost of the generated SN. Experiment 2 is to observe the performance fluctuation during the iterations of IEA to validate the philosophy of IEA. Experiment 3-5 are used to check the effects that three factors have on the performance of IEA, including QoS strictness of the incoming requirements, the criteria of sorting requirements before the iterative enhancement, and the ratio between NC/UC of candidate services.

6.2 Experiment 1: Performance w.r.t. Number of Requirements

The objective of this experiment is to compare the performance of the three SNP algorithms (SCA, RGA and IEA). Based on the same candidate services set, the algorithms execute to look for the optimal SN for different amounts of customer requirements (n=10, 20,..., 100). There are five performance indicators in the comparison, i.e., the number of atomic services in SN, number of service nodes in SN, number of edges in SN, total cost of SN, and the execution time of algorithms.

The results are shown in Figure 4. Compared with SCA and RGA, IEA shows better performance. Specifically, in terms of the same amounts of personalizes requirements, the number of service nodes, atomic services and edges in the SNs generated by IEA are all comparatively smaller than the ones by SCA and RGA, implying that IEA produces SNs with smaller sizes and complexity. This also results in lower cost. In addition, IEA exhibits better execution efficiency than SCA and RGA: the execution time of IEA increases at a very low speed due to the sorting and iterative

http://www.hipore.com/ijsc
 enhancement strategy, but the one of SCA is directly proportional to the amount of requirements. The comparison between SCA and RGA proves the advantage of requirement grouping: although the improvement degree on the size and complexity of SN are not quite large, the execution time decreases obviously. It is also seen that all the five indicators increase along with the increasing amount of customer requirements, i.e., more requirements result in larger and more complex SN with more cost, and consequently, more execution time to plan the SN.

This conclusion proves that, traditional one-requirement-oriented service composition approaches are overburdened in the mass customization scenario, but our iterative enhancement strategy behaves better by generating a smaller SN at lower cost and higher efficiency.

6.3 Experiment 2: Performance Fluctuation during the Iterations of IEA

In this experiment we observe the performance fluctuations during the iterations of IEA for 100 personalized requirements. There are totally 100 times of iterations during the execution, and each iteration deals with one requirement. Figure 5 shows the results.

![Figure 5. Performance fluctuations during IEA's iterations](http://www.hipore.com/ijsc)

From Figure 5(a) we see that, the number of service nodes, atomic services and edges in the SN all increase in a step-wise manner, and the growth rate becomes increasingly slow. This is in accordance with the IEA philosophy, i.e., those requirements that row in the forefront of the queue could cover more other requirements but they are seldom mutually covered, therefore more service node and atomic services are imported to satisfy them; but for the subsequent requirements, they could be satisfied by the existing SN with a higher probability so that less services and edges are to be enhanced to the exiting SN.

Average reusability refers to the average number of requirements that each atomic service participates in. From Figure 5(b) we can see that, the average reusability keeps increasing along with more and more requirements. This is because IEA tries to utilize the previously selected services to satisfy new-incoming requirements instead of importing new ones. But to note here is that it shows some sudden drops. This implies that the incoming requirement cannot be completely covered by previous ones, so that new services have to be imported into current SN. Even so, the average reusability increases linearly on the whole (see the cubic polynomial fitting in the figure).

6.4 Experiment 3: Performance w.r.t. QoS Strictness of Requirements (QS)

Different customers will raise different requirements, not only on the functionality but also on the QoS. Some requirements have more strict QoS constraints relative to the Willing To Pay (WTP) that the corresponding customers pay, and some others have more relaxed QoS constraints. Different QoS strictness of requirements will have some impact on the efficiency and performance of IEA.

Here we use three groups of requirements, each one having 20 distinct requirements. The functionality constraints and WTP are identical, and the only difference is that the requirements in Group 1 have quite relaxed QoS constraints, the ones in Group 3 have quite strict QoS constraints, and the ones in Group 2 have aligned QoS constraints. IEA is applied to the three groups, respectively, and Figure 6 shows the comparisons between the three groups of requirements which are labeled by Relaxed, Aligned and Strict, respectively.

Figure 6(a) is the comparison on the benefit and cost of the generated SN for the three groups of requirements. The SN for the Relaxed group yields the maximum benefit and minimum cost, and the one for the Strict group does the opposite. The reason is obvious: requirements with more strict QoS constraints require that the SN planning algorithm selects those services with higher QoS and consequently, higher UC; therefore, the total cost is higher. Because the WTP that the customers pay is fixed, the total benefit gained from the satisfaction of these strict requirements are meanwhile lower.

Figure 6(b) is a comparison of the complexity of the generated SN, including the number of service nodes, the number of atomic services, and number of edges. The SN for the Strict group has the maximal complexity, while the SN for the Relaxed group has the minimal complexity, but the difference is not that big, indicating that the QoS of requirements does not much affect the complexity of the generated SN.

http://www.hipore.com/ijsc
Figure 6. Performance comparisons among three groups of requirements having different QS

6.5 Experiment 4: Performance w.r.t. Sorting Criteria of Requirements

In IEA, the first step is to sort the requirements in terms of the potential benefit in the descending order, so that the requirements that could bring more benefit will be dealt with preferentially. Actually, there are other sorting criteria, such as sorting by WTP of the requirements (i.e., requirement with the highest WTP is given top priority), and sorting randomly (i.e., the order by which these requirements are dealt with is not considered). This experiment tries to verify whether and how different sorting criteria have influences on the performance of the SN constructed by IEA.

From Figure 7(a) we see that the total benefits under the three sorting criteria are almost coincident, implying that the sorting criteria will not affect the cost-effectiveness of the constructed SN. However, from Figure 7(b) and (c) we see the difference in the complexity of the constructed SN (number of service nodes, the number of atomic services, and number of edges), and in most cases, sorting by PB reduces the complexity of the SN in a certain degree. Another conclusion is that sorting by WTP will result in the highest complexity, implying that "greedy" is not a good sorting criterion. WTP is the direct benefit gained from the satisfaction of a requirement, while PB is the sum of both direct and potential benefits, and sorting by PB will make the algorithm select those services that might cover more other requirements, instead of only one requirement, so that the reusability of the selected services is improved. This experiment proves the effectiveness of the sorting by PB adopted in IEA.

6.6 Experiment 5: Performance w.r.t. Ratio between NC/UC of Candidate Services

In this experiment, we try to discover whether the ratio between the negotiation cost (NC) and the usage cost (UC) of candidate web services affects the performance of the SN constructed by IEA. In terms of different amounts of requirements (10, 20, ..., 100), IEA is executed under different NC/UC settings (1/4, 1, 4, 7, and 10). To note that, the sum of NC and UC of each candidate service remains the same, e.g., NC=10, UC=40, NC/UC=1/4; NC=25, UC=25, NC/UC=1; and so on. Figure 8 shows the comparisons on the total profit and cost of the constructed SN.

From the figure we find that, with the ever increase of NC relative to UC, the benefit brought by the SN is likely to inflate and the total cost of the SN is likely to decrease. This confirms the advantage of the iterative enhancement strategy, i.e., during the execution of ORSC in each iteration, the higher NC (and correspondingly, the smaller UC) will

http://www.hipore.com/ijsc
make the search tend to reuse those selected services because the added cost is only UC; and if UC is comparatively larger than NC, the significance of service sharing among multiple requirements becomes more trivial, so that the algorithm ties to look for new services instead of reusing existing ones, and consequently, the total cost increases and the profit decreases.

facilitating the two varieties are "multi-source parameters" and "compound service node", respectively. The core algorithm for Service Network Planning (SNP) problem is IEA, which considers the similarities and correlations of the functionality and QoS constraints among multiple requirements, and uses an iterative enhancement strategy to continuously extend an existing SN. Compared with two traditional approaches, i.e., SCA and RGA, a set of experiments validate the superiority of IEA.

A significant future work is to consider the growth of an existing SN. There are some common features in the crowd requirements, and such features are slowly evolving, so an SN should also follow the evolution and make changes on it. Key issues include: (1) how to identify the evolution of crowd requirements and behaviors; and (2) how to map it into the structure changes of an SN.

8. ACKNOWLEDGMENT
Research works in this paper are supported by the National Natural Science Foundation (NSF) of China (No. 61033005, 61272187, 61472106).

9. REFERENCES


http://www.hipore.com/ijsc
Authors

Zhongjie Wang received his PhD in 2006 from the School of Computer Science and Technology, Harbin Institute of Technology (HIT), China. He is now a professor of HIT. His main research interests include services computing, mobile and social networking services, and software architecture.

Nan Jing is now a master student in the School of Computer Science and Technology, Harbin Institute of Technology (HIT), China. Her research topic is mass customization of services.

Fei Xu got his master degree from the School of Computer Science and Technology, Harbin Institute of Technology (HIT), China in 2013. He is now a software engineer in Baidu Inc. His research interest is service networking and service composition.

Xiaofei Xu received his PhD in 1989 from the School of Computer Science and Technology, Harbin Institute of Technology. Currently, he is a professor and director of the Research Center of Intelligent Computing for Enterprises and Services (ICES). He is the assistant president of HIT. He has published four books and more than 200 papers. His current research interests include services computing, enterprise computing, data engineering and data mining.
AN ANALYSIS AND COMPARISON OF CLOUD DATA CENTER ENERGY-EFFICIENT RESOURCE MANAGEMENT TECHNOLOGY

[1] Zhihui Lu, 2 Soichi Takashige, 2 Yumiko Sugita, 2 Tomohiro Morimura, and 2 Yutaka Kudo

1School of Computer Science, Fudan University, Shanghai, China,
2IT Management Research Dept., Yokohama Research Laboratory, Hitachi, Ltd., Japan.
lzh@fudan.edu.cn, {zhihui.lu,soichi.takashige,ksyumiko.sugita.yf,tomohiro.morimura.kee,yutaka.kudo.sn
@hitachi.com}

Extension 6011 at SCC Industry 2013

Abstract

Nowadays, cloud data centers began to support more and more popular online services such as web search, e-commerce, social networking, video on demand and software as a service (SaaS), so the massive scale of data centers brings the challenges of energy efficiency. Therefore, the concept of an energy-efficient Green data center has been proposed. To build an energy-efficient cloud data center, cloud data center resource dynamic provisioning and consolidation technology is involved. In this paper, we first provide a survey of current industry and academic efforts on cloud data center energy-efficient management technology, focusing on the cloud data center resource dynamic provisioning technology and resource consolidation technology. We first focus on an analysis and comparison of cloud resource predictive dynamic provisioning technology. We analyze and discuss the main resource prediction methods and models, including basic models, feedback based models and multiple time series models. We describe the relationship between these categories as well as the characteristics of the models. After that, we analyze and compare Cloud resource reactive dynamic provisioning technology. And then we analyze cloud resource consolidation technology. Furthermore, we also give a prospect on cloud resource management technology standardization trends. Lastly, we analyze the prospective research direction.

Keywords: [Cloud Computing, IDC, Cloud Data Center, Virtual Machine-VM, Resource Consolidation, Energy-efficient management technology, Resource Prediction, Resource Management, VM Migration]

1. INTRODUCTION

With the arrival of the twenty-first century, Web technologies have matured increasingly with the introduction of multiple applications such as e-commerce, real-time communication, video on demand, and social networking and so on. These entities require high-speed networks to connect and guarantee the performance of the running applications. To meet the resource demands of these applications, plenty of Internet Data Centers (IDCs) are built around the world. Benefiting from Internet economies and recent developments in Web technologies, data centers have emerged as a backbone of today’s IT infrastructure to provision resources to multiple applications for satisfying their availability and performance requirements. The rise and rapid development of cloud computing virtualization technologies are bringing the traditional IDC a revolutionary change to Cloud data center. Server, storage and network virtualization are at the base of the flexibility, affordability and scalability of any cloud-based data center offering.

Meanwhile, with the increasing scale of Internet applications every year, the size of data center scales up in proportion to the service expansion, and data centers consume a great deal of power energy every year. The more hosts within a data center, the more power energy is consumed. It was reported that 2012 saw an increased focus on the data center industry from the media, public and government bodies, all concerned about its use of energy (DCD Intelligence 2014). There is a 19% increase in the amount of electricity consumed globally by data centers between 2011 and 2012 (DCD Intelligence 2014). The results from the 2013 annual DCD Intelligence (DCDi) Industry Census show that the rise in power for 2013 was over 7% (DCD Intelligence 2014). The facts tell us that such a dramatic energy consumption increase in cloud data center requires a scalable and dependable IT infrastructure comprising of servers, storage, network bandwidth, electrical Grid and cooling system.

In modern data centers, there are hundreds of thousands of hosts used to meet application demands. However, the data center resources are usually in low utilization. Based on the report of Data Center Efficiency Trends for 2014 from “Energy Manager Today” (Aaron R. 2013), in current data center, server utilization rates are typically very low, currently averaging in the 6–12 percent range. A completely idle server still draws 60 percent of its maximum power. While large data centers enjoy economies of scale by amortizing long-term capital investments over large number of machines, they also incur tremendous energy costs in terms of power distribution and cooling (Qi Zh., et al., 2013). In particular, it has been reported that energy-related costs account for approximately 12% of overall data center expenditures. For large companies like Google, a 3% reduction in energy cost can translate to over a million dollars in cost savings (Qi Zh., et al., 2013). On the other
hand, governmental agencies continue to implement standards and regulations to promote energy-efficient computing (Green Computing). As a result, reducing energy consumption has become a primary concern for today’s data center operators (Qi Zh., et al., 2013). The goal of the energy-efficient resource management is to dynamically adjust and minimize the number of active machines in a data center in order to reduce energy consumption while meeting the Service Level Agreements (SLA) of applications.

In the context of energy-efficient resource management, energy non-proportionality of IT resource, resource over-provisioning for near-peak performance should be considered by the resource management solutions. One of the simplest yet most effective approaches for reducing energy costs is to dynamically adjust the data center capacity by turning on needed machines and turning off unused machines, or to set them to a sleep state. This is supported by the evidence that an idle machine can consume as much as 50-60% of the power when the machine is fully utilized (Qi Zh. 2013). Unsurprisingly, a number of efforts are trying to leverage this fact to save energy by minimizing the number of active machines using a combination of server consolidation and dynamic capacity provisioning (Qi Zh. 2013):

Server consolidation aims at finding an assignment of workload to machines in order to minimize the number of used machines (Qi Zh. 2013). Server consolidation and storage consolidation with virtualization can help reduce the number of data center running servers, storage, desktop, and network devices to reduce complexity and make IT management simpler. Typically, server consolidation is achieved through (1) resource-aware workload scheduling and (2) dynamically adjusting workload placement using migration (Qi Zh. 2013). The former approach relies on the scheduler to find a good initial placement of workloads, whereas the second approach realizes that the initial workload placement can become sub-optimal over time, and improves workload placement using techniques such as VM migration.

On the other hand, dynamic capacity provisioning aims at dynamically controlling the number of active machines by switching machines on and off (or in and out of "sleep" state), based on various factors such as the workload arrival rate, workload performance requirement and the electricity price (Qi Zh. 2013). While over-provisioning the data center capacity can lead to sub-optimal energy savings, under-provisioning the data center capacity can cause significant performance penalty in terms of scheduling delay, which is the time a resource request has to wait before it is scheduled in the data center. In data centers, some application workload exhibits periodical nature. Sometimes workload is high and the resource is at high utilization, while other time the load is low and part of the resource is idle. It is typical that the load is orderly during some period. This introduces the technology of resource prediction. Many resource usage prediction algorithms have been exploited to predict some future parameters, such as system workload, the number of physical machines needed and so on. At the same time, we need to design reactive techniques, as the name suggests, referring to the methods that can quickly react to the abrupt fluctuation of the current system and applications status.

Energy saving technologies can be implemented through data center hardware and physical infrastructure, or upper resource management software, as shown in Figure. Some energy-aware hardware technology generally uses dynamic voltage and frequency scaling (DVFS), and many cooling infrastructure, such as temperature-aware air conditioner, has already been deployed. As for management software, the main concern is with the energy-efficient resource management. Cloud resource consolidation and dynamic resource provisioning focuses on energy-efficient cloud resource management field, where the main idea is to improve the utilization efficiency of the cloud system resource while meeting the Service Level Agreements (SLA) of applications. Baliga J, et al. (Baliga J, et al.,2011) present Green Cloud computing: an analysis of energy consumption in cloud computing. The analysis considers both public and private clouds, and includes energy consumption in switching and transmission as well as data processing and data storage. Buyya et al. (Buyya et al., 2010) proposed the GreenCloud project aimed at the development of energy-efficient provisioning of Cloud resources, while meeting QoS requirements defined by the SLAs established through a negotiation between Cloud providers and consumers. Beloglazov A., et al. (Beloglazov A., et al.,2011) give a good survey on the causes and problems of high power / energy consumption, and presents a taxonomy of energy-efficient design of computing systems covering the hardware, operating system, virtualization, and data center levels.

![Energy-ware Hardware](image)

Figure 1. Energy Saving Technology Architecture in Cloud Data Center

Consolidating many VM servers in a small number can contribute to reducing the cost of energy in a substantial way (Medina, V., et al., 2014). Consolidating cloud data center resources through virtualization strategy not only increases utilization without incurring additional risk but it will ultimately help avoid costly, unnecessary new resource construction of data centers. The consolidation mechanism is an example of power saving management through policies that handle the idle, standby, and sleep states. Consolidation

http://www.hipore.com/ijsc
can be used to minimize the number of currently active resources so that standby and sleep states can be used. Underused servers can be consolidated into fewer hosts and offloaded hosts can be placed into a sleep mode (Medina, V., et al., 2014). Therefore, it realizes energy saving during resource consolidation process, and an energy saving object can often be achieved through resource consolidation in turn.

Dynamic resource provisioning has a lot of potential for reducing power consumption in data centers. Most dynamic resource provisioning approaches can be categorized into two types: predictive (or proactive) and reactive (or control-theoretic). However, there are many challenges that hinder the successful deployment of dynamic resource provisioning. A common assumption in resource provisioning is that workload demand can be predicted (Gandhi A., et al., 2013). Therefore, proactive or prediction-based resource provisioning is required so as to deal with the periodic resource usage pattern. Unfortunately, this is not always the case. Even for long-term (daily) workloads with periodic trends, short-term fluctuations cannot be predicted based on periodicity (Gandhi A., et al., 2013). This observation suggests that purely predictive approaches might be insufficient for handling data center workloads. Ignoring this unpredictable portion can lead to a steep increase in response times. Further, another challenge in dealing with unpredictable workload demand is the possibility of load spikes (Gandhi A., et al., 2013). While load spikes, or abrupt changes in load, are not necessarily a daily occurrence in data centers, several instances of load spikes have been documented for web workloads. Important events, such as the September 11 attacks, earthquakes or other natural disasters, slashdot effects, Black Friday shopping, or sporting events, such as the Super Bowl or the Soccer World Cup, are common causes of load spikes for website traffic. Service outages or server failures can also result in abrupt changes in load caused by a sharp drop in capacity. Most of the above events cannot be predicted in advance. Therefore, some researchers propose reactive resource provisioning schemes.

The main contribution of this paper focuses on two aspects: 1. Cloud resource predictive and reactive provisioning method analysis and comparison. In this paper, we analyze and discuss the main cloud resource predictive and reactive methods. 2. Virtualized resource consolidation technology analysis. We analyze and compare the cloud virtualized resource consolidation schemes. Furthermore we analyze the characteristics and factors which impact the cloud resource consolidation.

The remainder of this paper is organized as follows: In Sec. 2, we make a detailed survey on current research on key technologies for building cloud virtualized data center and realizing cloud resource energy-efficient management. In Sec. 3, we introduce and describe the main cloud resource prediction models, and have an analysis and comparison on these models in Sec. 4. We analyze and compare cloud resource reactive provisioning schemes in Sec 5. Next, we analyze and compare cloud resource consolidation schemes in Sec 6. After that, we give a prospect for cloud resource management technology standardization trends in Sec 7. Finally, we conclude the paper and propose our future work in Sec 8.

2. Survey on Cloud Virtualized Data Center Resource Energy-efficient Management Technology

2.1 Survey on Cloud Data Center Resource Dynamic Provisioning Technology

Cloud computing has emerged as one of the most promising business models for the future. For a growing number of businesses, the journey to cloud computing starts with a private cloud implementation, and this mainly focuses on IaaS (Infrastructure as a Service). In recent years, a great number of IT enterprises have been committed to offering products related to this area.

Currently, there are four mainstream production cloud virtualization platforms from different companies: VMware®'s vSphere®, Microsoft®'s Hyper-V®, open source Xen® sponsored by Citrix®, and kernel-based KVM® supported by Red Hat®. The comparison of these four platforms is described in Table 1. Xen supports CPU DVFS with P-states and C-states (CPU sleep state) (Beloglazov A., et al., 2011). Xen also supports offline and live migration of VMs. KVM supports the S4 (hibernate) and S3 (sleep/stand by) power states. Similar to Xen, VMware supports host-level power management via DVFS. And VMware® Motion® enables live migration of VMs between physical nodes (Beloglazov A., et al., 2011). Currently, VMware’s vCenter® and Citrix’s XenServer® have begun to provide the power management function, and VMware Distributed Resource Scheduler (DRS) contains a subsystem called VMware Distributed Power Management (DPM) to reduce power consumption by a pool of servers by dynamically switching off spare servers, but they don’t support high level energy-efficient policy. The free Microsoft® Assessment and Planning (MAP) Toolkit can consolidate servers and implement some power management features based on Hyper-V. The four most popular open source cloud management platforms are OpenStack® (OpenStack, 2014), CloudStack® (Apache CloudStack, 2014), OpenNebula® (OpenNebula, 2014) and Eucalyptus® (Eucalyptus, 2014). These open source cloud management platforms have provided basic resource management function, but need to extend to support realizing energy-efficient management policy and function module. As a representative, OpenStack Neat (OpenStack Neat, 2014) is an extension to OpenStack implementing dynamic consolidation of Virtual Machines (VMs) using live migration. The major objective of dynamic VM consolidation is to improve the utilization of physical resources and reduce energy consumption by re-allocating VMs using live migration according to their real-time
resource demand and switching idle hosts to the sleep mode. The aim of the OpenStack Neat project is to provide an extensible framework for dynamic consolidation of VMs based on the OpenStack platform.

Table 1. Comparison of Main IaaS Management Products

<table>
<thead>
<tr>
<th>Company</th>
<th>Virtualization Product</th>
<th>Management Product</th>
<th>Main Management Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMware (VMware, 2014)</td>
<td>vSphere (ESX/ESXi)</td>
<td>vCenter</td>
<td>Integrated management</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Workload balancing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Resource optimization</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Priority settings/affinity rules</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Power management etc.</td>
</tr>
<tr>
<td>Microsoft (Microsoft, 2014)</td>
<td>Hyper-V</td>
<td>System Center</td>
<td>Comprehensive configuration</td>
</tr>
<tr>
<td></td>
<td></td>
<td>family</td>
<td>Multi-hypervisor support</td>
</tr>
<tr>
<td>Citrix (Citrix, 2014)</td>
<td>Xen</td>
<td>XenServer</td>
<td>Workload balancing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Memory optimization</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Power management etc.</td>
</tr>
<tr>
<td>Red Hat (Red Hat, 2014)</td>
<td>KVM</td>
<td>CloudForms</td>
<td>Application lifecycle mgmt.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Compute resource mgmt.</td>
</tr>
</tbody>
</table>

2.1.1 Survey on Cloud Data Center Resource Predictive Dynamic Provisioning Technology

In recent years, resource prediction techniques have been used in the cloud data center environment as a necessary pre-step to dynamic resource provisioning. As a supplement, it helps to manage resources more efficiently. Prediction techniques have been in existence long before and there are many advanced models. In cloud resource dynamic provisioning schemes, some prediction methods are totally based on these traditional models, but there are some novel prediction models transforming from traditional models and more suitable for cloud resources.

In terms of traditional prediction techniques, there are some mature models. Mikhail Dashevskiy et al. (Dashevskiy, M., et al., 2009) consider Fractional Autoregressive Integrated Moving Average (FARIMA) processes which provide a unified approach to characterizing both short-range and long-range dependence. Taylor J.W. et al. (Taylor, J.W., et al., 2007) study some short-term load prediction methods, including ARIMA modeling, periodic AR modeling, an extension for double seasonality of Holt-Winters exponential smoothing and a recently proposed alternative exponential smoothing formulation. Antti Sorjamaa et al. (Sorjamaa, A., et al., 2007) propose a global methodology for the long-term prediction of time series. This methodology combines direct prediction strategy and sophisticated input selection criteria to forecast future values. In (Taylor J.W., et al., 2006; Tan, J., et al., 2011), the authors present a prediction method based on PCA (principal component analysis).

With the development of cloud computing, prediction techniques are introduced in order to provide the virtualized resources more dynamically and efficiently. In recent years, a great number of prediction methods are used in the cloud. Besides these, we present a few well-performed models used in a data center environment. At present, most models are built based on time series analysis. Depending on the time series, we divide these prediction models into three categories: the first category is basic models and the second is feedback based prediction models. Both of these two categories are built based on single time series. The third category is for multiple time series prediction models. The classification chart is shown in Fig. 2.

![Classification of Resource Prediction Model](image)

First, we list some basic prediction models. In order to achieve efficient auto-scaling in the cloud, Nilabja Roy et al. (Roy, N., et al., 2011) develop a model-predictive algorithm for workload forecasting. They use a second order autoregressive moving average method (ARMA) filter to have a single look-ahead prediction.

Agustín C. Caminero (Caminero, A.C., et al., 2011) and Danilo Ardagna (Ardagna, D., et al., 2011) use an exponential smoothing model to forecast the workload of their cloud system, so as to implement resources provisioning. Zhenhuan Gong et al. (Gong, Z., et al., 2010) present a novel predictive elastic resource scaling (PRESS) scheme for cloud systems. This scheme leverages Fast Fourier Transform (FFT) and discrete-time Markov chain to forecast future demand. It can handle both cyclic and non-cyclic workload. Brian Gønter et al. (Gønter, B., et al., 2011) present a regression based method to predict the number of physical machines needed in some future time. Gong Chen et al. (Chen, G., et al., 2008) develop a prediction model named Sparse Periodic Auto-Regression. This model uses auto-regression and local adjustment to build the prediction model. We consider these models to be basic models, because these methods just use prediction techniques to get future values, without considering other information about application.

Qingjia Huang et al. (Huang, Q., et al., 2014) introduce a Prediction-based Dynamic Resource Scheduling (PDRS) solution to automate elastic resource scaling for virtualized cloud systems. Unlike traditional static consolidation or threshold-driven reactive scheduling, they both consider the dynamic workload fluctuations of each VM and the resource conflict handling problem. PDRS first employs an online resource prediction, which is a VM resource demand state
predictor based on the Autoregressive Integrated Moving Average (ARIMA) model, to achieve adaptive resource allocation for cloud applications on each VM. Then they propose the prediction-based dynamic resource scheduling algorithms to dynamically consolidate the VMs with adaptive resource allocation to reduce the number of physical machines. Extensive experimental results show that our scheduling is able to realize automatic elastic resource allocation with acceptable effect on SLAs.

In some resource management schemes, feedback control is added to the prediction models to improve resource allocation. Wesam Dawoud et al. (Dawoud, W., et al., 2011) present elastic VM architecture for cloud resources provisioning based on feedback control. In this architecture, each controller is designed to predict the next resource allocation based on the last allocation and consumption. Zhiming Shen et al. (Shen, Z., et al., 2011) present CloudScale, a system that automates fine-grained elastic resource scaling for multi-tenant cloud computing infrastructures. It employs online resource demand prediction and prediction error handling to achieve adaptive resource allocation. Jie Zhu et al. (Zhu, J., et al., 2011) present a dynamic allocation framework for Database-as-a-Service. In their prediction model, in order to alleviate underestimation, a small offset is added to the predicted value. Pradeep Padala et al. (Padala, P., et al., 2009) present AutoControl, a resource control system of multiple virtualized resources. In this system, there is a model estimator, which inputs past allocation, past performance and leverage autoregressive moving average (ARMA) model to achieve the future performance value. With the predicted performance value, optimizer module determines the resource allocation.

There is another kind of prediction method, which adopts multiple time series to build prediction models. Jian Tan et al. (Tan, J., et al., 2011), Liang, Jin et al. (Liang, J., et al., 2004) present multi-resource prediction models for resource sharing environments, like cloud, data center and so on. In their models, they not only consider the autocorrelation of a single resource, but also the cross correlation between different resources. Arjit Khan et al. (Khan, A., et al., 2012) present a multiple time series approach for workload characterization and prediction in the cloud. In their approach, it treats server workload data samples as multiple time series, and divides servers into different clusters. Then it characterizes the correlations in these clusters to predict for each cluster. Yexi Jiang et al. (Yexi, J., et al., 2011) present an online system to model and predict the cloud VM demand. They utilize two-level ensemble method to capture the characteristics of the high transient demand time series. The first level is a regression based ensemble that combines the results of different prediction models of the same VM type. The second level ensemble considers the relationship between different VM types, and utilizes their correlation to help improve the robustness of prediction.

In the on-demand cloud environment, web application providers have the potential to scale virtual resources up or down to achieve cost-effective outcomes. To address this challenge, Jiang jing et al. (Jiang, J., et al., 2013) propose a novel cloud resource auto-scaling scheme at the virtual machine (VM) level for web application providers. The scheme automatically predicts the number of web requests and discovers an optimal cloud resource demand with cost-latency trade-off. Based on this demand, the scheme makes a resource scaling decision that is up or down or NOP (no operation) in each time-unit re-allocation. Their experiment results demonstrate that the proposed scheme achieves resource autoscaling with an optimal cost-latency trade-off, as well as low SLA violations. On-demand resource provisioning is with great challenge in cloud systems. The key problem is how to know the future workload in advance to help determine resource allocation. There is various prediction models are developed to predict the future workloads and provide resources. The major problem of previous research is that they assume that application workload has static pattern. In the paper (Feifei Zh., et al., 2013), we present a Pattern Sensitive Resource Provisioning Scheme, named PSRPS. It can recognize application workload dynamic patterns and choose suitable prediction models for prediction online. Besides, when there is misadjustment in prediction models, PSRPS can switch prediction models or adjust the parameters of the model by itself to adaptively to guarantee prediction accuracy. This paper extends paper (Feifei Zh., et al., 2013)’s work.

Gandhi A., et al.(Gandhi A.,et al., 2014) propose a new cloud service, Dependable Compute Cloud (DC2), that automatically scales the infrastructure to meet the user-specified performance requirements. DC2 employs predictive Kalman filtering to automatically learn the (possibly changing) system parameters for each application, allowing it to proactively scale the infrastructure to meet performance guarantees. DC2 is designed for the cloud - it is application-agnostic and does not require any offline application profiling or benchmarking. Their implementation results on OpenStack using a multi-tier application under a range of workload traces demonstrate the robustness and superiority of DC2 over existing rule-based approaches.

2.1.2 Survey on Cloud Data Center Resource Reactive Dynamic Provisioning Technology

In general, predictive approaches are very successful when dealing with periodic or seasonal workloads. However, these approaches fail when the workload is bursty and unpredictable, or when the workload demand suddenly increases: it is clearly hard to predict what will happen in the future when demand is bursty and future arrivals are unknown(Gandhi A., et al., 2013). Therefore, some studies also propose cloud resource reactive dynamic provisioning technology. In this situation, we need to consider how to reduce the setup time of VM in a short time with a reactive way, especially for public cloud, and some researchers use...

http://www.hipore.com/ijsc

36
VM image pre-copying method to reduce VM launching time. At the same time, we need to avoid resource over-provisioning for workload peak time, resulting in resource waste and excessive energy consumption.

Wang et al. (Peijian W., et al., 2010) employ a reactive feedback mechanism to manage the power-performance tradeoff in multi-tier systems. The authors use DVFS along with capacity provisioning to react to degradation in observed response times. Wang et al. leverage queueing theoretic results to help guide their system. However, the queueing models employed by Wang et al. are much simpler, and do not take setup time into account.

Gandhi A., et al. (Gandhi A., et al., 2012) introduce a dynamic capacity management policy, AutoScale, that greatly reduces the number of servers needed in data centers driven by unpredictable, time-varying load, while meeting response time SLAs. AutoScale scales the data center capacity, adding or removing servers as needed. AutoScale has two key features: (i) it autonomically maintains just the right amount of spare capacity to handle bursts in the request rate; and (ii) it is robust not just to changes in the request rate of real-world traces, but also request size and server efficiency.

In the paper (Guo, T., et al., 2014), Tian Guo et al. describe Seagull, a system designed to facilitate cloud bursting by determining which applications should be transitioned into the cloud and automating the movement process at the proper time. Seagull optimizes thebursting of applications using an optimization algorithm as well as a more efficient but approximate greedy heuristic. Seagull also optimizes the overhead of deploying applications into the cloud using an intelligent pre-copying mechanism that proactively replicates virtualized applications, lowering the bursting time from hours to minutes.

As the popularity of cloud computing continues to rise, more and more applications are being deployed in public clouds. To conserve provisioning costs while achieving performance objectives, clients should automatically scale up and down applications deployed in the cloud to match changing workload demands. The cloud provider, on the other hand, should attempt to consolidate load onto highly utilized physical machines, in order to reduce wasted power consumption. Tighe, M et al. (Tighe, M. et al., 2014) propose a new algorithm combining both the automatic scaling of applications with dynamic consolidation of virtual machines, in order to meet the goals of both the cloud client and provider. This allows the cloud provider to make better use of their infrastructure, and reduce power consumption.

Reducing the energy footprint of warehouse-scale computer (WSC) systems is key to their affordability, yet difficult to achieve in practice. The lack of energy proportionality of typical WSC hardware and the fact that important workloads (such as search) require all servers to remain up regardless of traffic intensity renders existing power management techniques ineffective at reducing WSC energy use. Lo, D. et al. (Lo, D., et al., 2014) present PEGASUS, a feedback-based controller that significantly improves the energy proportionality of WSC systems, as demonstrated by a real implementation in a Google search cluster. PEGASUS uses request latency statistics to dynamically adjust server power management limits in a fine-grain manner, running each server just fast enough to meet global service-level latency objectives. In large cluster experiments, PEGASUS reduces power consumption by up to 20%.

Swama et al. (Mylavarapu, S., et al., 2010) came up with a better capacity planning algorithm that “could ensure that it plans for peak usage but do not provision for it”. They modeled the problem as a stochastic optimization problem with the objective of minimizing the number of servers while considering two important constraints a) stochastic nature of workloads and b) minimizing the application SLA violations. Padala et al. (Padala, P., et al., 2009) presented a resource control system that automatically adapted to dynamic workload changes to achieve application SLOs. It was a combination of an online model estimator and a novel multi-input, multi-output (MIMO) resource controller, in which the former one captured the complex relationship between application performance and resource allocations, while the latter one allocated the right amount of multiple virtualized resources to achieve application SLOs.

2.2 SURVEY ON CLOUD DATA CENTER RESOURCE CONSOLIDATION TECHNOLOGY

Whether predictive provisioning approaches or reactive provisioning approaches, we need deploy VM resources on physical server to contain applications. Consolidating multiple applications on a single physical server can solve issues related to low utilization, however, how to efficiently and accurately perform server consolidation at enterprise datacenter level is still an unsolved research problem that faces significant technical challenges (Lei L., 2014) including how to accurately measure and characterize an application’s resource requirements, how optimally to distribute the virtual machines hosting the applications over the physical resources, and how to avoid performance interference among the virtual machines collocating in the same physical machines (Lei L., 2014).

As a first step toward enabling energy efficient consolidation, Srikantaiah S. et al. (Srikantaiah, S., et al., 2008) study the inter-relationships between energy consumption, resource utilization, and performance of consolidated workloads. The study reveals the energy performance trade-offs for consolidation and shows that optimal operating points exist. They model the consolidation problem as a modified bin packing problem and illustrate it with an example.

Verma et al. (Verma, A. et al., 2008) investigated the problem of dynamic placement of applications in virtualized systems, while minimizing power consumption and meeting the SLAs. To address the problem, the authors proposed the pMapper application placement framework. It consists of
three managers and an arbitrator, which coordinates their actions and makes allocation decisions. Performance Manager monitors the behavior of applications and resizes the VMs according to the current resource requirements and SLAs. Power Manager is in charge of adjusting hardware power states and applying DVFS. Migration Manager issues instructions for VM live migration to consolidate the workload. Arbitrator has a global view of the system and makes decisions about new placements of VMs and determines the VM reallocations necessary to achieve a new placement.

Effective consolidation of different applications on common resources is often akin to black art as application performance interference may result in unpredictable system and workload delays. In the paper (Lei L., et al., 2013), Lu lei et al. consider the problem of fair load balancing on multiple servers within a virtualized data center setting. They especially focus on multi-tiered applications with different resource demands per tier and address the problem on how to best match each application tier on each resource, such that performance interference is minimized. To address this problem, they propose a two-step approach.

First, a fair load balancing scheme assigns different virtual machines (VMs) across different servers; this process is formulated as a multi-dimensional vector scheduling problem that uses a new polynomial-time approximation scheme (PTAS) to minimize the maximum utilization across all server resources and results in multiple load balancing solutions. Second, a queueing network analytic model is applied on the proposed min-max solutions in order to select the optimal one. Experimental results show that the proposed mechanism is robust as it always predicts the optimal consolidation strategy.

In particular, the consolidation of multiple customer applications onto multi-core servers introduces performance interference between colocated workloads, significantly impacting application QoS. To address this challenge, Ripal Nathuji et al. (Nathuji, R., et al., 2010) advocate that the cloud should transparently provision additional resources as necessary to achieve the performance that customers would have realized if they were running in isolation. Accordingly, they have developed Q-Clouds, a QoS-aware control framework that tunes resource allocations to mitigate performance interference effects. Q-Clouds uses online feedback to build a multi-input multi-output (MIMO) model that captures performance interference interactions, and uses it to perform closed loop resource management. In addition, they utilize this functionality to allow applications to specify multiple levels of QoS as application Q-states. For such applications, Q-Clouds dynamically provisions underutilized resources to enable elevated QoS levels, thereby improving system efficiency. Experimental evaluations of their solution using benchmark applications illustrate the benefits: performance interference is mitigated completely when feasible, and system utilization is improved by up to 35% using Q-states.

Dejan et al. describe the design and implementation of Deep-Dive, a system for transparently identifying and managing interference (Novakovic, D., et al., 2013). DeepDive successfully addresses several important challenges, including lack of performance information from applications, and large overhead of detailed interference analysis. They first show that it is possible to use easily-obtainable, low-level metrics to clearly discern when interference is occurring and what resource is causing it. Next, using realistic workloads, they demonstrate that DeepDive quickly learns about interference across colocated VMs. Finally, they show DeepDive’s ability to deal efficiently with interference when it is detected, by using a low-overhead approach to identifying a VM placement that alleviates interference.

Dejan et al. (Vasić, N., et al., 2012) also propose DejaVu – a framework that deals with interference by estimating an “interference index”. Their approach to dealing with performance interference on the virtualized hosting platform recognizes the difficulty of pinpointing the cause of interference, and the inability of cloud users to change the hosting platform itself to eliminate interference. DejaVu uses a pragmatic approach in which it probes for interference and adjusts to it by provisioning the service with more resources.

Large-scale datacenters (DCs) host tens of thousands of diverse applications each day. However, interference between colocated workloads and the difficulty to match applications to one of the many hardware platforms available can degrade performance, violating the quality of service (QoS) guarantees that many cloud workloads require. While previous work has identified the impact of heterogeneity and interference, existing solutions are computationally intensive, cannot be applied online and do not scale beyond few applications. Christina D. et al. (Christina D. et al., 2013) present Paragon, an online and scalable DC scheduler that is heterogeneity and interference-aware. Paragon is derived from robust analytical methods and instead of profiling each application in detail; it leverages information the system already has about applications it has previously seen. It uses collaborative filtering techniques to quickly and accurately classify an unknown, incoming workload with respect to heterogeneity and interference in multiple shared resources, by identifying similarities to previously scheduled applications. The classification allows Paragon to greedily schedule applications in a manner that minimizes interference and maximizes server utilization. Paragon scales to tens of thousands of servers with marginal scheduling overheads in terms of time or state.

The cost and benefit of virtual machine migration for resource consolidation are also taken into consideration for some research such as (Qiang H., et al., 2011; Verma, A., et al., 2010; Haikun L., et al., 2011). In (Qiang H., et al., 2011), the authors studied the power consumption of virtual machine live migration. They conducted experiments and

http://www.hipore.com/ijs
the results showed that the power influence of migration to the original server decreases when the CPU usage of the migrated VM increases, but to the destination server, the influence is stable. In (Verma, A., et al., 2010), researchers found that live migration required a significant amount of space CPU on the source server, and the amount of CPU required would increase with an increase in the number of active pages of the VM being migrated. Besides, a colocated VM would impact a VM being migrated by taking away resources from the physical server. Haikun L. et al. (Haikun L., et al., 2011) thoroughly analyze the key parameters that affect the migration cost from theory to practice. They construct two application-oblivious models for the cost prediction by using learned knowledge about the workloads at the hypervisor (also called VMM) level. This should be the first kind of work to estimate VM live migration cost in terms of both performance and energy in a quantitative approach. There exist unconsidered conflicting factors impacting the VM migration process such as load volume, power consumption and resource wastage. Sallam, Ahmed et al. (Sallam, A., et al., 2014) consider the migration process as a multi-objective problem where the objectives are typically non-commensurable. Therefore, they propose a novel migration policy consolidated by a new elastic multi-objective optimization strategy to evaluate different objectives (including migration cost) simultaneously, and to provide the flexibility for manipulating different cases. They have tested the proposed policy through an extensive set of simulation experiments using CloudSim, and the results ensure the efficiency of our policy to control the system performance by adjusting the migration objectives to suit various workload consolidation situations.

Beloglazov A., et al. (Beloglazov A., et al., 2012) propose a novel adaptive heuristics for dynamic consolidation of VMs based on an analysis of historical data from the resource usage by VMs. The proposed algorithms significantly reduce energy consumption, while ensuring a high level of adherence to SLA. They validate the high efficiency of the proposed algorithms by extensive simulations using real-world workload traces from more than a thousand PlanetLab VMs.

3. INTRODUCTION AND DESCRIPTION OF CLOUD RESOURCE PREDICTIVE DYNAMIC PROVISIONING TECHNOLOGY MODELS

In recent years, resource predictive dynamic provisioning technologies have been used in cloud data center environments, as a necessary pre-step to realize cloud resource dynamic provisioning on predicted demand. We have listed the main cloud resource prediction models in Figure 2. We introduce and describe the main mechanism of these models in this section.

3.1 Basic Prediction Models

Most classical methods for prediction are based on time series analysis and there also are some advanced models, such as ARMA models, ARIMA models, and state-space models etc. Because of the computational complexity of these methods, most prediction models adopted in cloud are simplified. Here we will introduce some basic prediction methods.

Niblajba Roy et al. (Roy, N., et al., 2011) develop a model-predictive algorithm for workload prediction that is used for resource autoscaling in cloud. In their strategy, they use a second order autoregressive moving average method (ARMA) filter. The equation for the filter used is given as

\[
\lambda(t + 1) = \beta \times \lambda(t) + \gamma \times \lambda(t - 1) + (1 - \beta + \gamma)(\lambda(t - 2))
\]  

(1)

The values for the variables \( \beta \) and \( \gamma \) are given by the values 0.8 and 0.15 respectively.

This ARMA model is a simple case and there are some deficiencies. Firstly, they use history data directly to model the workload pattern, without a pre-processing, which may result in jitter in prediction. Secondly, assigning the variables in the model to specified values will affect the prediction accuracy.

Danilo Ardagna et al. (Ardagna, D., et al., 2011) propose capacity allocation techniques able to coordinate multiple distributed resource controllers working on distributed cloud sites. They leverage exponential smoothing to predict the local arrival rate. The detailed description of the ES-based prediction model is as follows. Suppose the time scale \( T_i \), at sample \( t \), the ES model predicts the local arrival rate at \( T_i \) steps ahead, \( \lambda^i_k(t + T_i) \), as a weighted average of the last sample \( \lambda^i_k(t) \) and of corresponding predicted sample \( \lambda^i_k(t) \), that is equal to:

\[
\lambda^i_k(T_i) = \frac{1}{T_i} \sum_{t=1}^{T_i} \lambda^i_k(t)
\]  

(2)

\[
\lambda^i_k(t + T_i) = \lambda^i_k(t) \times \lambda^i_k(t) + (1 - \lambda^i_k(t)) \lambda^i_k(t), t > T_i
\]  

(3)

Where \( \lambda^i_k(T_i) \) is the initial predicted value and \( 0 < \lambda^i_k(t) < 1 \) is the smoothing factor at current sample \( t \) related to the site \( i \) and the class \( k \) that determines how much weight is given to each sample. They obtain a dynamic ES model by re-evaluating the smoothing factor \( \lambda^i_k(t) \) at each prediction sample \( t \). the smoothing parameter is defined as the absolute value of the ratio of the smoothed error \( \lambda^i_k(t) \) to the absolute error, \( \lambda^i_k(t), \lambda^i_k(t) = \frac{A_k^i(t)}{\lambda_k^i(t)} \) (Trigg, D., et al., 1967).

In this model, the evaluation of the smoothing factor is critical in that the selection of this factor has a direct impact on the algorithm accuracy. For a better fitting to the system, they leverage dynamic ES model, so they have to re-evaluate its smoothing factor frequently.

http://www.hipore.com/ijsc
Brian Guenter et al. (Guenter, B., et al., 2011) present a method to predicate the number of physical machines needed in some future time. In the paper, they present ACES, an automated server provisioning system that aims to meet workload demand while minimizing unmet load demand. The ACES has key components: a load predictor for proactive provisioning and an optimizer framework. The predictor is based on regression analysis to predict load in the near future. The method is as follows:

\[ m_0 = [a_1, ..., a_p] \begin{bmatrix} m_{-p} \\ \vdots \\ m_{-1} \end{bmatrix} \]  

(4)

The future demand \( m_0 \) is predicted by taking the dot product of coefficient vector \( a \) with a vector of past demand values, \( m(t) \). Where \( (a_1, ..., a_p) \) denote the \( p \) coefficients of a weighted liner predictor and are computed by solving the least-squares problem.

This method directly develops the prediction model with historical data, without considering the workload patterns. It is simple, but may have some improvement on accuracy.

Gong Chen et al. develop a prediction algorithm in (Chen, G., et al., 2008). In order to achieve energy-aware server provisioning, they exploit short-term load forecasting method. In this method, let \( y(t) \) be the stochastic periodic time series under consideration, with a specified time unit. It can represent the parameters measured at regular time intervals. Suppose the periodic component has a period of \( T \) time units. The value of \( y(t) \) in terms of all previous measurements as:

\[ y(t) = \sum_{k=1}^{n} a_k y(t - kT) + \sum_{j=1}^{m} b_j y(t - j) \]  

(5)

\[ \Delta y(t - j) = y(t - j) - \frac{1}{n} \sum_{k=1}^{n} y(t - j - kT) \]  

(6)

There are two parts in the model. The part with parameters \( a_k \) does periodic prediction—it is an auto-regression model for the value of \( y \) over a period of \( T \). The part with parameters \( b_j \) gives local adjustment, meaning the correlations between \( y(t) \) and the values immediately before it. The integer \( n \) and \( m \) are their orders, respectively. It is called a SPAR (Sparse Periodic Auto-Regression) model and has a good expansibility to multiple seasonal components.

In this model, there is an assumption that the workload is periodic. In some systems, however, workload is not cyclic. So this assumption limits its use.

Zhenhuan Gong et al. (Gong, Z., et al., 2010) present a novel predictive elastic resource scaling (PRESS) scheme for cloud systems. This approach leverages light-weight signal processing and statistical learning algorithms to achieve online predictions of dynamic application resource requirements. For workloads with repeating patterns, PRESS derives a signature for the pattern of historical resource usage, and uses that signature in its prediction. PRESS uses signal processing techniques to discover the signature, or decide that one does not exist. It employs a Fast Fourier Transform (FFT) to calculate the dominant frequencies of resource-usage variation in the observed load pattern. If there are multiple dominating frequencies, PRESS picks the lowest dominating frequency \( f_d \). With \( f_d \), PRESS derives a pattern window size of \( Z \) samples: \( Z = (1/f_d) \times r \) where \( r \) denotes the sampling rate. It then splits the original time series into \( Q = [W/Z] \) pattern windows: \( P_1 = \{l_1, ..., l_{l_2}\} \), \( P_2 = \{l_2+1, ..., l_{l_2}\} \), .... \( P_l = \{l_{(l-1)2}+1, ..., l_{2\times l}\} \). PRESS evaluates the similarity between all pairs of different pattern windows \( P_i \) and \( P_j \) to discover whether it includes repeating patterns. If all pattern windows are similar, PRESS treats the resource time series as having repeating behavior, and uses the average value of the samples in each position of pattern windows to make its prediction. However, for applications don’t have repeating patterns, PRESS uses a discrete-time Markov chain with a finite number of states to build a short-term prediction.

This prediction model can conduct a prediction without assuming any prior knowledge, like the repeating pattern, period. But sometimes the least frequency is not the best choice. Besides, if there is no repeating pattern, it leverages the Markov chain to give a prediction, and the accuracy will be impacted.

### 3.2 Feedback Based Prediction Models

Despite the potential for basic prediction models to perform very well in some systems, in order to guarantee application performance, there have been some prediction models combined with control theory. With performance feedback information, the predicted values can be remedied adaptively to obtain higher accuracy.

Jie Zhu et al. (Zhu, J., et al., 2011) propose a dynamic resource allocation framework (DRAF) for Database-as-a-Service, in which there is a prediction model. In their model, time series is assumed to be represented as trend plus noise, i.e. \( y(t) = T_{\mu}(t) + e_{\sigma}(t) \), where \( T_{\mu}(t) \) is the trend and \( e_{\sigma}(t) \) is the noise. \( T_{\mu}(t) \) is obtained through regression analysis, and \( e_{\sigma}(t) \) is assumed to have a normal distribution unless additional information about noise becomes available. A trend pattern is represented as \( T_{\mu}(t) = b_0 + b_1 t + b_2 t^2 \). The least-squares method is used to abstract patterns. Noise is described in two parameters: the mean value \( \mu \) and the standard deviation \( \sigma \).

According to the normal distribution table, \( \mu+\sigma = 1.96\sigma \) can cover over 95% of data. Therefore, \( e_{\sigma}(t) = \mu+1.96\sigma \). In every interval, predictor (a component) adjusts the model based on the analyzer’s output. Lastly, a small offset is added to the predicted curve.

This prediction model leverages a simple method to predict the workload. It neglects the cyclical component in the time series, which in some applications is a major characteristic. Meanwhile, the remedy of this method is coarse.
Zhiming Shen et al. (Shen, Z., et al., 2011) present CloudScale, a system that employs online resource demand prediction and prediction error handling to achieve adaptive resource allocation in the cloud. Online resource demand prediction frequently makes over- and under-estimation errors. Over-estimation may result in resource waste, but it will not impact the application performance. On the contrary, under-estimation will result in SLO violations for insufficient resource allocated to application. In CloudScale, the basic prediction method is based on the model presented in (Barham, P., et al. 2003). The main work to alleviate under-estimation errors is prediction error correction. Here, we introduce how the prediction error correction module works. The module incorporates both proactive and reactive approaches to handle under-estimation errors. CloudScale uses an online adaptive padding scheme to avoid under-estimation errors by adding a small extra value to the predicted resource demand. When the application suffers SLO violations during resource under-estimation, it uses fast under-estimation correction to solve the problem. A simple solution is to immediately raise the resource cap to the maximum possible value. Another method is to raise the resource cap gradually by multiplying the current resource cap by a ratio a>1 until the error is corrected.

Pradeep Padala et al. (Padala, P., et al., 2009) present a resource control system AutoControl, a control system of multiple virtualized resources. In this system, a model estimator inputs past performance and past allocation to get the performance target in the future, and then the optimizer computes the resource needed. For every control interval, the model estimation re-computes a linear model that approximates the quantitative relationship between the resource allocations to application a (uj) and its normalized performance (yj) around the current operating point. The relationship is represented by the following auto-regression moving average (ARMA) model:

\[ \hat{y}_j(k) = a_1(k) \hat{y}_j(k-1) + a_2(k) \hat{y}_j(k-2) + b_0^T(k) u_j(k) + b_1^T(k) u_j(k-1) \]  

Where \( a_1(k) \) and \( a_2(k) \) capture the correlation between the application’s past and present performance, and \( b_0(k) \) and \( b_1(k) \) are vectors of coefficients capturing the correlation between the current performance and recent resource allocations. With the performance target, optimizer determines the resource allocations required.

This model reflects the concept of control theory. The resource need in the future is determined both by application’s performance and past allocation. In this model, the prediction is simple, and the key is an optimizer which transforms the performance target to the resource needed.

### 3.3 Multiple Time Series Prediction Models

In the multiple time series prediction, prediction models are developed based on both autocorrelation of a single resource and cross correlation between different resources. Considering the cross correlation can help to obtain a more accurate prediction model on the premise that there indeed exist some relationships among these resources.

Liang, Jin et al. (Liang, J., et al., 2004) propose a multi-resource prediction model (MModel) in a distributed resource sharing environment. This model uses both the autocorrelation and the cross correlation to achieve higher prediction accuracy. The autocorrelation characterizes the statistical relationship of some resource at different times, and the cross correlation characterizes the statistical relationship between two resources. The multi-resource prediction model is as follows:

\[ \hat{x}_k = \sum_{i=1}^{p} a_i x_{k-1} + \sum_{i=1}^{q} b_i y_{k-i} \]  
\[ \hat{y}_k = \sum_{i=1}^{p} c_i y_{k-1} + \sum_{i=1}^{q} d_i x_{k-i} \]

This means the resource value \( x_t \) is predicted from not only the history value \( x_t \) itself but the resource value \( y_t \). The same is true for the prediction of \( y_t \). Here \( p \) is the auto-regression order and \( q \) is the cross regression order.

In the case of MModel, model fitting is similar to that of an AR model. In general, this model can achieve a higher accuracy compared with AR models considering only autocorrelation. There are some assumptions that the mean and autocorrelation of the resource are time invariant. Because of the cross correlation, there is another assumption that the cross correlation is also time invariant. All these assumptions limit the use of this model.

Jian Tan et al. (Tan, J., et al., 2011) propose two ways, from microscopic and macroscopic perspectives, to predict the resource consumption for data centers by statistically characterizing resource usage patterns. The first approach focuses on the usage prediction for a specific node, and the second approach is based on Principal Component Analysis (PCA) to identify resource usage patterns across different nodes. Auto-regression models for a VM: let \( X_j^0(t) = (x_j^0(t), x_j^0(t)) \) denote the stationary multivariate time series obtained by preprocessing the raw data. \( X_j^0(t) \) denotes the CPU usage and \( X_j^0(t) \) denotes the memory usage. They apply standard AR models to characterize data. It relies on both CPU and memory series for prediction, using

\[ X_j^1(t) = b_{j,11} X_j^0(t-1) + b_{j,12} X_j^0(t-1) + \beta_j e_j^1(t) \]  
\[ X_j^2(t) = b_{j,11} X_j^0(t-1) + b_{j,12} X_j^0(t-1) + \beta_j e_j^2(t) \]

Where \( e_j^1(t), e_j^2(t) \) are white noise(\( \mathcal{WN}(0, \sigma^2) \)). The coefficients are estimated with the Yule-Walker method (Peter, J.B., et al., 2002). PCA based prediction across nodes: this method uses principal component analysis to identify the resource usage patterns across the whole data center, and then conducts AR model prediction on a subset of principal components that account for most of the variability in the measurements. After forecasting resource usages for the
small number of principal components, it will then map the predictions along these identified principal components to the resource usage observation space as the prediction.

The whole prediction strategy has two levels. The first one characterizes the usage pattern for each VM using an AR model. The higher level uses PCA to identify the principal components, and with this subset to forecast the whole system resource consumption.

Arijit Khan et al. (Khan, A., et al., 2012) present a multiple time series approach for workload characterization and prediction in the cloud. This approach firstly groups servers into some co-clusters by a co-clustering method. In the workload prediction, the first step is to analyze the temporal correlation of server workload at different time granularity, and determine the optimal interval for developing a prediction model. Step 2 is to identify the predictable co-clusters from the original co-clusters. Step 3 is to construct the prediction model. In order to predict workload changes for those predictable co-clusters, they divide them into prediction groups and a prediction model can be developed for each of these groups. Paroli et al. model the workload variations across all co-clusters in a group as a continuous-time Markov process (Paroli, R., et al., 2002). Using a Hidden Markov Model (HMM), they define some parameters. Once the state transition and observation probabilities have been estimated, given the current observation at any time t, the probability of observing o_t+j at time t+1 can be calculated as:

\[ P(H_t | o_t) \sum_{H_{t+1}} P(H_{t+1} | H_t) P(o_{t+1} | H_{t+1}) \]  

With this probability estimation, it can predict the most likely observation at time t+1 as

\[ o'_{t+1} = \arg \max_{o_t} P(o_t | o_t), o_t \in O \]  

Note that the next observation represents a possible combination of which co-clusters in a prediction group would appear. Therefore, if a co-cluster is predicted to appear in the next time interval, we can predict accordingly the discretized workload levels of all associated servers.

To improve prediction accuracy, this approach develops a model that captures workload patterns along both temporal and spatial dimensions. Developing a prediction model specialized for a group can characterize the series patterns more precisely. While all the prediction work is used for the predictable co-cluster, there is little concern for others without predictability.

Yexi Jiang et al. (Yexi, J., et al., 2011) present an online temporal mining system called ASAP, to model and predict cloud VM demands. It uses a two-level ensemble method to capture the characteristics of the high transient demands time series. The first level is a regression based ensemble that combines the results of different prediction models of the same VM type. The weights of these methods will be updated by calculating the relative error. The models used in this level include moving average, auto-regression, artificial neural network, support vector machine, and gene expression programming. The second level ensemble considers the relationship between different VM types, and utilizes their correlation to help improve the robustness of prediction. With the correlation information between time series to develop prediction model helps to alleviate the disturbance of the “noisy” data in real cloud service scenario. Suppose \( \text{cov}_{ij}^{(t)} \) denotes the covariance between resource types i and its \( j \)th correlated resource, the post-processed predicted demand for type i at time t should be

\[ \hat{u}_i^{(t)} = \frac{\sum_{j=1}^{k} \text{cov}_{ij}^{(t-1)} s_{ij} \hat{P}_k^{(t)}}{\sum_{j=1}^{k} s_{ij} \text{cov}_{ij}} \]  

Where \( s_{ij} = \hat{v}_i - \hat{v}_j \) denotes the difference of scale between two time series and \( k \) is the number of strong correlated time series.

This model considers both the autocorrelation of each VM type and the cross correlation between different types to improve the prediction accuracy. Meanwhile, in the first level, it integrates many prediction models to characterize the pattern of series, which leads to greater computational capability at some degree.

4. ANALYSIS AND COMPARISON ON CLOUD RESOURCE PREDICTIVE DYNAMIC PROVISIONING MODELS

4.1 CLASSIFICATION

In section 3, we divide the cloud resource prediction models into three categories: basic models, feedback based models and multiple time series models. From the description above we find whether feedback models or multiple time series models, their most fundamental work of prediction are based on basic models. So we can depict the relationships of these categories as follows, shown in Fig. 3.

**Figure 3. Relationships between three cloud resource prediction models**

In some systems that do not require high accuracy, basic models are enough for prediction. At present, in order to obtain a more efficient resource provisioning scheme, we need do some extra work on the prediction model, lest insufficient resource allocation leads to performance degradation, so feedback based models and multiple time series rise depending on basic models.

http://www.hipore.com/ijsc
4.2 Comparison and Analysis

We analyze and compare all of the models in each category in Table 2, 3, and 4 respectively. With respect to each category, we list different metrics to depict the models in it.

4.2.1 Basic Prediction Models

In Table 2, except for the FFT based model, their core methods for prediction are auto-regression. In the prediction area, auto-regression is a fundamental but widely used method. The AR model is a special case of the ARMA model. In (Peter, J.B., et al., 2002), it shows the whole theories about different prediction models, including the ARMA model. The AR model is simple and the accuracy is acceptable for some systems. The key issue of this method is estimation parameters, and (Peter, J.B., et al., 2002) shows some approaches for it. In the models here, they all leverage the least square method to estimate parameters. For the ES model, the key is to evaluate $\gamma(t)$ precisely, which influences the prediction accuracy directly. The problems with these models are that they are suitable for short-term prediction, but for long term prediction, they cannot guarantee accuracy. This is the same for the FFT based model. If there is a repeating pattern in the time series, it can predict the whole future values in a pattern window. However, if it fails to find a repeating pattern, it will adopt Markov chain to predict, which just give a short term prediction.

4.2.2 Feedback Based Prediction Models

The main feature of feedback based prediction models in Table 3 is to automatically provide resources for application and avoid SLOs violations. In order to guarantee SLOs, the first thing is adding an offset to the predicted value. Secondly, with feedback information, schemes rebuild the prediction models in every control interval. The new prediction models consider the past performance of applications relying on the past allocation, and make some adjustments. Thus, it can reflect the latest status of application better. In general, feedback based models can achieve better performance than basic models.

<table>
<thead>
<tr>
<th>Prediction models</th>
<th>Core method</th>
<th>Parameter calculation</th>
<th>feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARMA model(Roy, N., et al., 2011)</td>
<td>Auto-regression</td>
<td>specified</td>
<td>Simple; accuracy will be impacted by specified parameters</td>
</tr>
<tr>
<td>Exponential Smoothing based model(Ardagna, D., et al., 2011)</td>
<td>Exponential smoothing</td>
<td>$\gamma(t) = \frac{A_i(t)}{E(t)}$</td>
<td>$\gamma(t)$ estimation is critical</td>
</tr>
<tr>
<td>FFT based model(Gong, Z., et al., 2010)</td>
<td>Signature based method; Markov chain method</td>
<td>FFT</td>
<td>A model without any prior knowledge</td>
</tr>
<tr>
<td>Line prediction(Gaenter, B., et al., 2011)</td>
<td>Auto-regression</td>
<td>Least square method</td>
<td>Simple; regardless of the characteristics of time series, The combination of auto-regression and local adjustment</td>
</tr>
<tr>
<td>Sparse periodic auto-regression(Chen, G., et al., 2008)</td>
<td>Auto-regression</td>
<td>Least square method</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Prediction model</th>
<th>Feedback information</th>
<th>Error correction</th>
<th>feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>CloudScale (Shen, Z., et al., 2011)</td>
<td>FFT based model</td>
<td>SLO violations</td>
<td>adaptive padding; raising the resource cap</td>
<td>Handle under-estimation errors</td>
</tr>
<tr>
<td>DRAF(Zhu, J., et al., 2011)</td>
<td>Series=trend +noise</td>
<td>Different between the predicted and observed values</td>
<td>Addition of a small offset</td>
<td>Lower the risk of SLA violations</td>
</tr>
<tr>
<td>AutoControl(Padala, P., et al., 2009)</td>
<td>ARMA model</td>
<td>Past performance</td>
<td>No</td>
<td>Auto-adaptive to resource changes to achieve SLOs.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Core method</th>
<th>Cross correlation</th>
<th>feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-resource prediction (Tan, J., et al., 2011)</td>
<td>ARMA model</td>
<td>Different resources series</td>
<td>Consider the usage correlation between different resources</td>
</tr>
<tr>
<td>MModel(Liang, J., et al., 2004)</td>
<td>ARMA model</td>
<td>Different resources series</td>
<td>Consider the usage correlation between different resources</td>
</tr>
<tr>
<td>Group-level approach (Khan, A., et al., 2012)</td>
<td>Hidden Markov model</td>
<td>Different co-clusters’ series</td>
<td>Use spatial correlations to filter measurement noise</td>
</tr>
<tr>
<td>ASAP(Yexi, J., et al., 2011)</td>
<td>Some models cooperatively</td>
<td>Different VM types’ series</td>
<td>Mitigate the disturbance of the “noisy” data</td>
</tr>
</tbody>
</table>

4.2.3 Multiple Time Series Prediction Models

In Table 4, the first two models consider the correlations between different resources because in some systems a resource usage will influence some other resource’s usage. So they leverage this characteristic. But the premise is that the correlation between resources indeed exists, otherwise, it will not help to improve the prediction accuracy. In the last two models, the purpose of introducing correlation is to alleviate the influence of “noisy” data. In the process of building prediction model, if there are much “noisy” data, the model may not characterize the pattern of the series.

http://www.hipore.com/ijs
accurately. As for the introduction of cross correlation, we need to evaluate the necessity, because it will complicate the models.

In order to maximize resource utilization and meanwhile guarantee application SLOs, almost all resource dynamic provisioning schemes first adopt prediction techniques, especially for long-term demand forecast. Some new prediction models, such as feedback based models or multiple time series models perform better than the basic prediction model. Sometimes, we should balance the accuracy and complexity of the models and pick the most suitable one. High accuracy is not always required because different systems have different service requirements.

4.3 A BASIC PROCESS OF PREDICTION MODELING AND DYNAMIC RESOURCE PROVISIONING SCHEME

Here, we first describe a general procedure of prediction modeling which is a reasonable method widely used. Then we present an outline of cloud resource dynamic provisioning schemes based on the prediction model.

4.3.1 Time Series Modeling

In [Peter, J.B., et al.,2002], a general method is listed to build a prediction model. Although not all prediction methods do so, it gives us an overview on prediction.
- Plot the series and examine whether there is
  (a) a trend
  (b) a seasonal component
  (c) any apparent sharp changes in behavior
  (d) any outlying observations
- Remove the trend and seasonal components to get stationary residuals.
- Choose a model to fit the residuals.
- Forecasting will be achieved by predicting the residuals and then inverting the transformations to arrive at forecasts of the original series.

4.3.2 The Outline of Resource Predictive Dynamic Provisioning Scheme

Based on the discussion above, a reasonable resource predictive provisioning scheme should include the following steps: modeling, forecasting, error correction, performance analysis, re-modeling, as shown in Fig. 4. When we build a prediction model based resource dynamic provisioning scheme, we need not to follow these steps strictly, and we can adjust some intermediate modules flexibly.

Cloud resource reactive dynamic provisioning technology handles excess demand by computing additional capacity at short time scales (e.g., the number of core, amount of memory, or the number of servers), in response to excess workload that is beyond the predictive workload, i.e., the difference between the actual workload and forecast based workload. Summarized from the various research directions on reactive dynamic provisioning technology, we can conclude some of the key objectives with wide public concerns, shown in Figure 5.

Figure 5. Main Objectives of Cloud Resource Reactive Dynamic Provisioning Technology

To handle a broader spectrum of possible changes in workload, including unpredictable changes in the request size and server efficiency, Gandhi A., et al. introduce the AutoScale policy (Gandhi A., et al., 2012). While AutoScale-- addresses a problem that many others have looked at, it does so in a very different way. Whereas prior approaches aim at predicting the future request rate and scaling up the number of servers to meet this predicted rate, which is clearly difficult to do when request rate is, by definition, unpredictable, AutoScale-- does not attempt to predict future request rate. Instead, AutoScale-- demonstrates that it is possible to achieve SLAs for real-world workloads by simply being conservative in scaling down the number of servers: not turning servers off recklessly.

This article makes the following contributions.
- They overturn the common wisdom that says that capacity provisioning requires “knowing the future load and planning for it,” which is at the heart of existing predictive capacity management policies. Such predictions are simply not possible when workloads are unpredictable, and, they furthermore show they are unnecessary, at least for the range of variability in our workloads. They demonstrate that simply provisioning carefully and not turning servers off recklessly achieves better performance than existing policies that are based on predicting current load or over-provisioning to account for possible future load.
- They introduce our capacity inference algorithm, which allows us to determine the appropriate capacity at any point of time in response to changes in request rate, request size and/or server efficiency, without any knowledge of these quantities. They demonstrate that AutoScale, via the capacity inference algorithm, is robust to all forms of changes in load, including unpredictable changes in request size and unpredictable degradations in server speeds, within the range of our traces.

http://www.hipore.com/ijsc
Problem Analysis: AutoScale really takes a fundamentally different approach to dynamic capacity management than has been taken in the past. First, AutoScale does not try to predict the future request rate. Instead, AutoScale introduces a smart policy to automatically provision spare capacity, which can absorb unpredictable changes in request rate. Existing reactive approaches can be easily modified to be more conservative in giving away spare capacity so as to inherit AutoScale’s ability to absorb unpredictable changes in request rate. But the problem is when the heavy workload comes suddenly, we really need to start up more new VMs, and we need consider how to decrease the set up time for new VMs.

Therefore, we can see that the VM set up time is a big concern in cloud resource reactive dynamic provisioning methods to handle workload spike or cloud burst. Some researchers have used pre-copy VM image methods to solve this problem to some extent.

Gandhi A., et al. also present SoftScale, a practical approach to handling load spikes in multi-tier data centers without having to over-provision resources (Gandhi A., et al., 2013). SoftScale works by opportunistically stealing resources from other tiers to alleviate the bottleneck tier, even when the tiers are carefully provisioned at capacity. SoftScale is especially useful during the transient overload periods when additional capacity is being brought online. Importantly, SoftScale can be used in conjunction with existing dynamic server provisioning policies, such as AutoScale.

The key insight of Tian Guo et al. (Guo, T, et al., 2014) is that occasional pre-copying of virtual disk snapshots of a few overload-prone applications can significantly reduce the cloud bursting latency, since only the incremental delta of the disk state needs to be transferred to reconstruct the image in the cloud. Their work examines the impact of judiciously choosing the candidate applications for such pre-copying. They have developed a system called Seagull to address the above challenges; Seagull automatically detects when resources in a private cloud are overloaded, decides which applications can be moved to a public cloud at lowest cost, and then performs the migrations needed to dynamically expand capacity as efficiently as possible.

Seagull supports both horizontally and vertically scalable applications for cloud bursting. It also allows flexible policies to be specified in terms of which applications to periodically pre-copy into the cloud. By automating these processes, Seagull is able to respond quickly and efficiently to workload spikes, allowing the data center to be safely operated at a higher utilization level.

The decision of when to trigger a cloud burst involves monitoring one or more metrics and using a threshold on them. Depending on the scenario, Seagull can use system-level metrics (such as CPU utilization, disk/network utilization or memory page fault rate) or application-level metrics such as response time. They assume that the system administrator makes a one-time decision on which metrics are relevant and the corresponding thresholds. In case of system-level metrics, the desired metrics can be monitored at the hypervisor-level, and for application-level metrics, we assume the presence of a monitoring system such as Ganglia that supports extensions to monitor any application metric of interest. Once an overload warning is triggered, Seagull can use either an optimal ILP based algorithm or a greedy heuristic to decide which applications to move to the cloud.

Problem Analysis: Seagull work is that occasional pre-copying of virtual disk snapshots of a few overload-prone applications can significantly reduce the cloud bursting latency, since only the incremental delta of the disk state needs to be transferred to reconstruct the image in the cloud. The problem is how to signature and recognize which applications have overload-prone feature and then take efficient pre-copy action.

In AGILE, Hiep Nguyen, et al. (Nguyen, H., et al., 2013) also use dynamic VM cloning to reduce application startup times. By combining the medium-term resource demand prediction with the black-box performance model, AGILE can predict whether an application will enter the overload state and how many new servers should be added to avoid this. The dynamic copy-rate scheme completes the cloning before the application enters the overload state with minimum disturbance to the running system. AGILE is light-weight: its slave modules impose less than 1% CPU overhead.

Problem Analysis: AGILE can predict whether an application will enter the overload state and then take the measure of dynamic cloning, but in the situation of short-term workload fluctuating, the medium-term resource demand prediction method will not be efficient.

In CloudScale solution (Shen, Z., et al., 2011), when applying the predictor to the resource scaling system, they found that the resource scaling system needs to address a set of new problems in order to reduce SLO violations. First, online resource demand prediction frequently makes over- and under-estimation errors. Over-estimations are wasteful, but can be corrected by the online resource demand prediction model after it is updated with true application resource demand data. Under-estimations are much worse since they prevent the system from knowing the true application resource demand and may cause significant SLO violations. Second, co-located applications will conflict when the available resources are insufficient to accommodate all scale-up requirements.

CloudScale provides two complementary under-estimation error handling schemes: 1) online adaptive padding and 2) reactive error correction. Their approach is based on the observation that reactive error correction alone is often insufficient. When an underestimation error is detected, an SLO violation has probably already happened. Moreover, there is some delay before the scaling system can figure out the right resource cap. Thus, it is worthwhile to perform proactive padding to avoid under-estimation errors.
Problem Analysis: the prediction error correction module: When the prediction error correction module performs online adaptive padding that adds a dynamically determined cushion value to the predicted resource demand in order to avoid under-estimation errors. The problem is how to determine the suitable amount of this value to avoid resource over-provision.

6. ANALYSIS AND COMPARISON ON CLOUD DATA CENTER RESOURCE CONSOLIDATION TECHNOLOGY

The development of virtualization technology has made great contribution to today’s IDC: it can effectively help the IDC to improve its overall utilization of IT resources, as well as provide greater flexibility for IDCs. However, obviously, this requires a higher level of resource consolidation techniques than ever. Resource consolidation aims at finding an assignment of workload to machines in order to minimize the number of used machines. Server consolidation and storage consolidation with virtualization can help reduce the number of data center servers, storage, desktop, and network devices to reduce complexity and make IT management simpler. As mentioned above, typically, resource consolidation is achieved through (1) resource-aware workload scheduling and (2) dynamically adjusting workload placement using migration (Qi Zh. 2013).

Main Objectives

- Resource Utilization
- Energy-saving
- Migration Cost
- User SLA / Application SLO
- Performance
- Interference
- Other
- Pre-defined Rule

Figure 6. Main Objectives of Cloud Resource Consolidation Technology

The workload scheduling of virtual machines is one of the most important consolidation issues in the virtualized cloud data center management field. It aims at obtaining the mapping between virtual machines and physical machines. More specifically, the further resource consolidation is the dynamically adjusting workload placement using migration. A virtual machine can be migrated to another host if the server begins to experience overload, failures or for energy saving purpose. Migration is a tool for load balancing, dealing with hardware failures, consolidating resource, or reallocating resources. In addition, the whole migration process should be adaptive to the dynamic changes of the overall operating environment, and make timely adjustment.

Summarized from the various research directions on cloud virtualized resource consolidation issues, we can conclude some of the key objectives with wide public concerns, shown in Figure 6.

- Resource Utilization. Server virtualization and consolidation is an attractive and effective technology to improve the average utilization rates of IT resources which are currently in the 15-20% range (Van, H. N., et al., 2009). It enables smaller resource allocations than physical machine, which hence potentially benefits data centers by allowing several applications to make use of a physical machine. Many research efforts such as (Nakada, H. N., et al., 2009; Hirofuchi, T., et al., 2010; Gupta, R., et al., 2008) viewed this objective as the most basic and fundamental one. Commonly, they took the virtual machine requirements and physical machine size as input, considering resources like CPU, and memory, to compute for a solution with least physical machines.

- Energy-saving. Energy costs represent a significant fraction of a datacenter’s budget and this fraction is expected to grow continuously in coming years, which has become an increasingly important concern for many businesses. We focus on energy-efficient resource management technology analysis in this paper. It is recognized that consolidation of VM services workloads and dynamic VM migration based on virtualization techniques show a great opportunity for increased server utilization rate and power decrease (Petrucci, V., et al., 2009). VM migration provides mechanism to realize energy-saving through moving applications running on several underutilized servers to a reduced number of highly used servers. Several recent papers focus on this aspect, e.g. (Rao, L., et al., 2010; Jung, G., et al., 2010; Petrucci, V., et al., 2009; Minghong L., et al., 2013). Most of them proposed their own power model as well as workload model to evaluate the current state, and work on this energy optimization problem.

- Migration Cost. Undoubtedly, virtual machine live migration enhances the flexibility in data centers, and gives the possibility for the re-location of VMs and re-distribution of resources. But some researchers also emphasis that the action of “live migration” is definitely not cost-free, on the contrary, the end-to-end performance, power consumption and thermal impacts are significant during the process (Qiang H., et al., 2011; Frnak, et al., 2011). Based on this fact, many recent researches such as (Quaenter, B., et al., 2011; Jung, G., et al., 2010; Shrivastava, S., et al, 2011) begin to take migration cost into account and view it as a secondary goal besides the others. The evaluation indicator of this might be migration times or size of VM to be migrated. Some logical policies or special user requirements may also be taken into consideration when executing VM migration. For example, a user might require that two of their virtual machines never run on the same physical host so one will remain available even if a physical host fails (Hyser, C., et al., 2007), or try to deploy a set of VMS in separate physical nodes so as not to compete over the same resources (Tsakalozos, K, et al. 2010). These may be translated into some placement constraints, such as sets of virtual machines that must always/never be on the same host, as well as lists of permitted/forbidden hosts for certain virtual machines.

User SLA/Application SLO. The applications running inside the virtual machine are much more important than the
VM itself. Therefore, energy-efficient resource consolidation technology must not be at the expense of the users’ SLA and applications’ SLO. What’s more, it is quite difficult to satisfy service-level objectives (SLOs) of applications on shared infrastructure, as application workloads and resource consumption patterns change over time (Padala, P., et al. 2009). Therefore, cloud providers have to deal with the power-performance tradeoff-minimizing energy consumption while meeting consumer SLA. To solve this problem, research such as (Mylavarapu, S., et al. 2010; Padala, P., et al. 2009; Van, H. N., et al. 2009; Petrucci, V. et al. 2009) focused on the complex relationship between application performance and resource re-allocations (including VM migration), and further determined the new mapping.

Performance Interference. Despite the benefits of virtualization, including its ability to slice a PM well in terms of CPU and memory space allocation, performance isolation is far from perfect in these environments. As cloud providers continue to utilize virtualization technologies in their systems, this can become problematic. In particular, the consolidation of multiple customer applications onto multicore servers introduces performance interference between collocated workloads, significantly impacting application QoS. Specifically, a challenging problem for providers is identifying (and managing) performance interference between the VMs that are co-located at each PM. Effectively dealing with performance interference is challenging for many reasons.

First, the IaaS provider is oblivious to its customers’ applications and workloads, and it cannot easily determine that interference is occurring.

Second, interference is complex in nature and may be due to any server component (e.g., shared hardware cache, memory, I/O).

Further, interference might only manifest when the co-located VMs are concurrently competing for hardware resources. Some approaches have struggled to improve the performance interference issues (Lei L., et al. 2013; Nathuji, R., et al. 2010; Novakovic, D., et al. 2013; Vasić, N., et al. 2012; Christina D. et al. 2013). Some systems may attempt to ensure the workload on each VM is within a small tolerance of the other workloads on the same physical hosts, or may attempt to avoid congestion through live migration (Yi Zh., et al. 2009; Gerofi, B., et al. 2010).

Above lists some major objectives in the cloud resource consolidation scheme. These objectives are not mutually exclusive, though with possible conflicts, and can be somewhat combined together so as to achieve more comprehensive and better results. Some recent research has made effort towards addressing this multi-objective virtual machine placement problem and has made some progress, e.g. (Günter, B., et al. 2011; Jung, G., et al. 2010; Xu, J., et al. 2010; Salam, A., et al. 2014).

7. PROSPECT ON CLOUD RESOURCE MANAGEMENT TECHNOLOGY STANDARDIZATION TREND

As a growing number of cloud computing products and solutions emerged, some potential problems have been exposed. For example, users may worry about the vendor lock-in, in other words, it might be impossible to move service from one cloud to another. Or, the different management method and process offered by each cloud provider may cause interaction problems. Therefore, it becomes a must to establish some powerful standards, in order to unify the multitude of cloud vendors. Some international standardization organizations begin to work towards international standards of cloud computing resource management, but most of them don’t consider energy-efficient resource management.

DMTF Open Cloud Standards Incubator (DMTF, 2009) established 3 whitepapers in 2009 on use cases and reference architecture as they relate to the interfaces between a cloud service provider and a cloud service consumer. The goal of the Incubator is to define a set of architectural semantics that unify the interoperable management of enterprise and cloud computing. These papers summarize the core use cases, reference architecture, and service lifecycle. These building blocks will be used to specify the cloud provider interfaces, data artifacts, and profiles to achieve interoperable management. After Open Cloud Standards Incubator, DMTF has announced the normal cloud resource management standard (DMTF, 2011) in 2011: the Cloud Infrastructure Management Interface (CIMI). CIMI is a self-service interface for infrastructure clouds, allowing users to dynamically provision configure and administer their cloud usage with a high-level interface that greatly simplifies cloud systems management. The specification standardizes interactions between cloud environments to achieve interoperable cloud infrastructure management between service providers and their consumers and developers, enabling users to manage their cloud infrastructure use easily and without complexity. However, they do not consider energy-efficient management standardized interface, but energy-saving policies can be inserted as a flexible strategy.

Two of the recent representative drafts from IETF are on the subject of Virtual Resource Operations & Management in the Data Center (IETF, 2011.7), and Policies and dynamic data migration in DC (IETF, 2011.6). The former describes the problem of operational and management challenges that virtualization brings in the (carrier) data center as an enabler of new technologies such as self-provisioning and elastic capacity and related benefits of consolidation, reduced total cost of ownership, and energy management. While the latter one describes some examples of the policies and dynamic information that need to migrate with VM, the influence if they are not migrated with VM, the problems that need to be considered with migration.
policies and dynamic information. It also describes some existing network management protocols standardized by IETF and the advantages and disadvantages of them for operating policies and dynamic information migration respectively. When the operators carry out VM migration to realize resource consolidation, the goal of this standard draft is to justify that it is necessary to make effort on policy and dynamic information migration together with VM migration for large virtualized Data Center.

8. CONCLUSION AND FUTURE WORK

Cloud computing as a new service model has attracted more and more attention. In the industry, there have been some successful examples, such as Google® AppEngine®, Microsoft Azure®, IBM® Blue Cloud®, Amazon® EC2® and S3® etc. Managing data center resources at large scale in both public and private clouds is quite challenging. Power is an expensive resource in data centers. Unfortunately, a lot of power in data centers is actually wasted. One of the big reasons for this waste is low server utilization (Gandhi A., et al., 2013). Servers in data centers are often left “always on”, leading to only 10-30% server utilization (Gandhi A., et al., 2013). In fact, some related reports tell that the average data center server utilization is only 18% despite years of deploying virtualization aimed at improving server utilization. From a power perspective, low utilization is problematic because servers that are on, while idle, still utilize 60% or more of peak power (Gandhi A., et al., 2013). Further, servers that are on necessitate support from other IT infrastructure including the cooling, networking, and storage systems. Thus, idle servers also lead to indirect power consumption. Given the importance of reducing power consumption in data centers and the fact that data centers have low server utilization, we conclude the resource dynamic provisioning and consolidation solution is a key technology trend for building energy-efficient cloud data center. In this paper, we first make a survey of virtualized cloud data center technology, including current industry efforts and academic efforts, focusing on cloud resource dynamic provisioning technology and cloud resource consolidation technology. Next, we analyze and discuss cloud resource prediction models, including basic models, feedback based models and multiple time series models. Furthermore, we describe the relationship between these categories and the cloud resource prediction models’ characteristics. We list and compare cloud reactive dynamic resource provisioning scheme. After that, we analyze and compare the cloud resource consolidation technology for energy-efficient cloud data center. Furthermore, we analyze the factors which impact the efficiency of VM consolidation.

Dynamic resource provisioning and efficient resource consolidation may lead to significant reduction of the cloud data center energy consumption, while meeting the SLA requirements. Efficient resource prediction is extremely important for cloud data centers comprising multiple computer and storage nodes. There have been several resource prediction methods used in cloud resource dynamic provisioning areas. These methods estimate application workload or resource demand at some time in the future, and provide appropriate resources to meet service demand in advance. The more accuracy in the prediction method, the better service the cloud data center can provide. At the same time, the reactive resource provisioning methods also have played an important role to complement the predictive provisioning method. VM set up time is still a challenge issue when the reactive resource provisioning method is used to handle workload fluctuating or spike. We have built a cloud data center resource management prototype system based on OpenStack and we implement a load prediction algorithm based on auto-regression and we also realize migration-based resource peak load shifting strategies to support resource consolidation. These studies have gained some achievement (Wei F. et al., 2012; Feifei Zh. et al., 2013; Jie B., et al., 2014).

Cloud resource consolidation takes dynamic server provisioning one step further and allows different application instances to be co-located on the same physical server (Gandhi A., et al., 2013). The basic idea is to consolidate the workload from several, possibly under-utilized, servers onto fewer servers through migration. Once a decision to migrate one VM is made, we need to determine its best destination to realize resource consolidation while avoiding performance interference. VM consolidation allows the unneeded servers to be turned off or become sleep, resulting in lower power consumption, and higher system utilization. Therefore, after VM consolidation, it is necessary to efficiently determine when and which physical machines should be switch off to save power, or switch back to the active stage for handling the new workload (Beloglazov A., et al., 2011). Data center operators should consider that frequently switching off and back servers increases the wear-and-tear cost of servers, which might also increase the risk of hardware failure. On the other hand, cloud provider is unaware of the application types when consolidating different workloads: different applications have different resource usage pattern. This makes it challenging to co-locate different application instances together on a physical server. For example, it might be more beneficial to co-locate a CPU-intensive application with an I/O-intensive application rather than co-locating two CPU-intensive applications. Identifying the resource usage patterns of different applications is often very difficult because of the heterogeneous nature of application workloads. There is also the concern of security risk, for instance, VM colocation might expose critical applications to security risks because of other suspicious and dangerous applications. VM colocation attack is an important issue to consider while carrying out VM consolidation.

http://www.hipore.com/ijsc
9. ACKNOWLEDGMENT
This paper work is based on the Fudan-Hitachi Innovative Software Technology Joint Laboratory project-cloud data center virtualized resource management system and Hitachi Yokohama Research Laboratory project-cloud data center resource scheduling management system. This work is also supported by 2014-2016 PuJiang Program of Shanghai under Grant No. 14PJ1431100.

10. REFERENCES

http://www.hipore.com/ijs
Policies and dynamic information migration in DCs, pp. 386-390. ACM.


Predictive vm consolidation on multiple resources: Beyond load balancing. In Quality of Service (IWQoS), 2013 IEEE/ACM 21st International Symposium on (pp. 1-10). IEEE.

Effective Resource and Workload Management in Data Centers, PhD Thesis, the College of William and Mary, Jan.


Towards energy proportionality for large-scale latency-critical workloads. In Proceeding of the 41st annual international symposium on Computer architecture (pp. 301-312). IEEE/Press.


Authors

Dr. ZhiHui Lu is an Associate Professor in School of Computer Science, Pudan University, China. Now he is also a senior researcher of IT Management Research Department, Yokohama Research Laboratory, Hitachi Ltd, Japan. His research interests are cloud computing, system resource management, and network multimedia technology. He is an expert member of China SOA and cloud computing standard work group led by CEST. He is also an academic alliance member of DMTF WS-Management Work Group.

Mr. Soichi Takashige is a researcher of IT Management Research Department, Yokohama Research Laboratory, Hitachi Ltd, Japan. His research interests are cloud data center resource management based on OpenStack. He is a team leader of OpenStack research team in YRL.

Ms. Yuniko Sugita is a chief staff researcher of IT Management Research Department, Yokohama Research Laboratory, Hitachi Ltd, Japan. Her research interests are cloud data center resource management and scheduling technology, and she is a Linux Architect.

Dr. Tomohiro Morimura is unit manager of IT Management Research Department, Yokohama Research Laboratory, Hitachi Ltd, Japan. His research interests are IT system management, virtual network and storage management, and IaaS cloud resource management technology.

Dr. Yutaka Kudo is general manager of IT Management Research Department, Yokohama Research Laboratory, Hitachi Ltd, Japan. His research interests are IT system management, service management, and cloud data center resource management technology, and PaaS Cloud management.

http://www.hipore.com/ijsc
MODELING AND ANALYSIS OF MOBILE PUSH NOTIFICATION SERVICES USING PETRI NETS
(Extended version of 7186 at SCC 2014)

Junhua Ding\textsuperscript{1, 2}
\textsuperscript{1) Dept. of Computer Science}
East Carolina University
Greenville, NC 27587
dingj@ecu.edu

Wei Song
Dept. of Computer Science
Nanjing University of Sci. & Tech.
Nanjing, China
wsong@njust.edu.cn

Dongmei Zhang\textsuperscript{2}
\textsuperscript{2) School of Computer Sciences}
China University of Geosciences
Wuhan, Hubei, China
jjielee@163.com

Abstract
Mobile push notifications are an important feature in mobile computing services and they have been widely implemented in mobile systems. However, it also brings the vulnerability of security and reliability to the system. Formal specification and verification is an effective approach for understanding the properties of mobile push notification services and ensuring quality of the system development. Due to the dynamic interaction, security and mobility properties in mobile computing services, formally modeling and analyzing them is a grand challenge. In this paper, we proposed an approach to model mobile computing services using a high level Petri nets and analyze them through combining formal verification and testing techniques. The dynamic interaction between users and service providers is modeled with the publisher subscriber architecture. The mobility is modeled with a connector that dynamically connects a user to its service provider based on user’s runtime environment, and the security is modeled with a threat model. The effectiveness of the approach is demonstrated with a case study of modeling and analyzing a mobile searching engine that is implemented with mobile push notifications services.

Keywords: mobile computing service; mobile push notification; formal specification; Petri net; testing

1. INTRODUCTION
Mobile push notifications are an important feature of mobile computing services and they have been widely implemented in mobile applications [1]. Mobile push notification services have become an important type of services to deliver contents to users. Many widely used commercial mobile applications, such as social network system Facebook, hangout system Tinder, instant messaging system WeChat, were implemented with mobile push notifications. Applications supporting special services such as weather service, travel service or traffic service were also implemented with mobile push notifications to deliver time sensitive and personalized contents to mobile users. The most important character of mobile push notifications is a service provider pushes real time notifications to subscribers based on their current contexts like their locations, status, and emotions. Mobile push notifications are fairly complex to implement due to several reasons: they have to be implemented based on a third party service such as Apple or Google mobile push notification services, they have to be implemented on secure communication protocols to ensure the security of the mobile communication, and they have to deal with the mobility issue and the dynamic connection between users and service providers. In order to understand mobile push notifications and build high quality mobile computing services, it is necessary to model them with a formal specification and analyze the model rigorously. In this paper, we proposed an approach to model mobile push notification services using a high level Petri net and analyze them through combining formal verification and testing techniques. The dynamic interaction between users and service providers is modeled with the publish-subscribe architecture. The mobility is modeled with a connector that dynamically connects a user to its service provider based on user’s runtime environment, and the security is modeled with a threat model to ensure the service is appropriately protected from attacks.

The effectiveness of the approach is demonstrated with a case study of modeling and analyzing a mobile searching system that is implemented with mobile push notifications services. The system was built on Amazon AWS, and the mobile push notification was supported by Amazon notification service and Google mobile push notification service [1][2]. We modeled the mobile push notification using Predicate Transition nets (PrT) – a type of high level Petri nets [13], and rigorously analyzed them using a model based testing tool that supports both formal analysis and software testing. Comparing to other methods, the PrT nets model developed in the approach is executable so that they support simulation and testing easily. More important, the model can be developed step by step based on simulation and analysis results during modeling process, and tests used for testing the model can be directly used for testing the implementation. Therefore, there is no gap between the
analysis technique using for testing models and the analysis technique using for testing the implementation.

The contributions of this research are summarized as follows: We proposed an approach for formally modeling and rigorously analyzing the mobile computing services, particularly the mobile push notification services. We addressed the challenge issues on mobility, security and dynamic interaction in modeling and analysis. The effectiveness of the approach is demonstrated through a case study of modeling and analyzing a mobile searching system. The approach can be used for modeling and analyzing similar mobile computing systems. The modeling process is explained via modeling the mobile searching system in the case study. A PrT net model of the mobile push notification implemented in the system is refined step by step with simulation and analysis results. The case study shows the analysis results-guided modeling approach is effective for understanding and developing formal models of complex systems. The analysis process is explained via analyzing the mobile searching service in the case study. The case study also shows the model based analysis is an easy to use and an effective approach for analyzing complex systems.

The rest of this paper is organized as follows: Section 2 presents the preliminaries for modeling and analyzing mobile push notification services. Section 3 introduces the approach for modeling and analyzing mobile push notification services in general. Section 4 describes a case study of modeling and analyzing a mobile searching system that is implemented with mobile push notification services. Section 5 reviews the related work. Finally, we outline our conclusions as well as future work in Section 6.

2. Preliminaries

In this section, we discuss the preliminaries for modeling and analyzing mobile computing services. The modeling language used in this paper is PrT nets, and the analysis approach is developed based on tool MISTA. We also discuss the architecture of mobile push notifications.

2.1 PrT Nets

Predicate/Transition (PrT) nets, a high level Petri net, are used for specifying software systems. The formal definition of PrT nets used in this paper is same as the one defined in [16], which restricted the predicate types as enumerable types

Definition 1 (PrT net). A PrT net is a tuple $(P, T, F, \Sigma, L, \phi, M_0)$, where:
1. $P$ is a finite set of predicates (i.e. first order places), $T$ is a finite set of transitions and $F$ is a flow relation (i.e. normal arcs). $(P, T, F)$ forms a directed net.
2. $\Sigma$ is a structure consisting of some sorts of individuals (i.e. constants) together with operations and relations.
3. $L$ is a labeling function on arcs.
4. $\phi$ is a mapping from a set of inscription formulae to transitions. $\phi(t)$ is built from variables, relations, operations and constants in $\Sigma$.
5. $M_0$ is the initial marking, where $M_0(p)$ is the set of tokens in place $p$. Each token is a tuple of constants in $\Sigma$.

Fig. 1 shows a PrT net model for the 5 dining philosophers’ problem, which includes transitions Pickup, and Putdown, places Phi, Chop and Down. Places Phi and Chop include tokens that are nature numbers representing philosophers or chopsticks, and each token in place Down includes represents a philosopher and his/her two chopsticks. Transition Pickup has two input places Phi and Chop, and one output place Down. The guard condition in transition Pickup is defined based on the relation between the tokens in place Phi and Chop. The guard condition in transition Putdown is defined based on the relation between the tokens in place Phi and Chop.

2.2 Model-Based Testing and MISTA

MISTA is a model-based testing tool for automated generation of executable test code in model level and program level. It uses function nets (a type of PrT nets extended with inhibitor arcs and reset arcs) [19] for specifying test models so that complete tests can be automatically generated. It also provides a language for mapping the elements in function nets to implementation constructs, which makes it possible to convert the model level tests into program level tests that can be executed against the system under test. MISTA includes several important components: model editor, model parser, simulator, reachability analyzer, test generator, and test code generator. MISTA support the step-by-step execution and random execution of a function net, and the execution sequences and token changing in each place are visualized for inspection. The test generator generates model level tests (i.e., firing sequences of the function net) according to a chosen coverage criterion such as transition coverage or state coverage. The tests are organized and visualized as a transition tree. MISTA supports a number of coverage criteria for test generation from function nets, including...
reachability graph coverage, transition coverage, state coverage, depth coverage, and goal coverage. Test code generator generates test code in a chosen target language like Java or C++ from a given transition tree [19].

2.3 Mobile Push Notifications

We explain the mobile push notifications using Amazon Simple Notification Service (SNS) [1] as an example since the system we discussed in the case study was implemented on Amazon SNS. We will discuss a representative mobile push notification service in details in section 3. Amazon SNS is a web service for managing the delivery of messages from publishers to subscribers. A publisher sends a message to a communication channel called a topic in SNS, and a subscriber who subscribed the topic will receive the message or notification via a communication protocol such as email or HTTP/S [1]. Amazon SNS sends the notification message to clients via one of the push notification services: Apple Push Notification Services (APNS), Google Cloud Messaging for Android (GCM), and Amazon Device Messaging (ADM). In order to use a push notification service, a mobile client needs to register itself for the service so that the service can maintain a connection to the client. When a mobile client registered a push notification service, the service returns a device token representing the registered device to Amazon SNS for registering the mobile client in SNS. Using the notification service credentials provided by a notification publisher, Amazon SNS can communicate with the mobile push notification service providers on behalf of the publisher. A publisher sends a message to a topic in Amazon SNS, and then Amazon SNS checks mobile devices according to the tokens it received. The notification message is sent to the push notification services, and each notification service pushes the message to the subscribed mobile clients [1].

One of application examples of mobile push notification is group online game, where game players connect to the online game provider using their mobile devices. While one is playing the game, actions like move and attack from each player appear on the screen right way. When one player switches from the game to another application while other players are still playing, it is necessary to notify the player when an action from others might be of interest to this one. Mobile push notifications can be used for the notification purpose. The publisher of the game application monitors game actions and finds one action may affect the player who just switched from the game to another application (i.e., the player is in no longer connected to the game), a push notification message is pushed to the player. First the publisher, who has sent its credentials to Amazon SNS and was authorized to access the SNS, sends the notification message to Amazon SNS. Then Amazon SNS, which has received the device tokens from all players, sends the messages to the push notification services. Finally, one of the push notification services such as APNS the mobile client has registered for pushes the notification message to the registered mobile device that the player is off from the game. A notification could be implemented in different formats like displaying a message, displaying an alert, badging the game app’s icon, or even playing a sound. The player may response the notification immediately or later.

3. Modeling and Analysis of Mobile Push Notification Services

In this section, we discuss how to model mobile push notification services using PrT nets, and how to analyze the model using model based testing technique.

3.1. Architecture

The mobile push notification services offered by Apple, Google and Amazon support the publish-subscribe (PS) style of interaction between publishers who produce content and subscribers who subscribe the content. The PS architecture supports the asynchronous interaction between publishers and subscribers, so that message exchange is still possible even one of them is not in active [14]. In order to support the asynchronous interaction, the architecture needs support content queuing and delivering. A GCM implementation includes three components: a GCM-enabled app running on an Android device, an application server that produces notification messages to the app, and a Google GCM server that is responsible for delivering messages from the application server [2]. Each application server has a Sender ID to identify itself, each app has an application ID for receiving appropriate messages, and a registration ID issued by GCM server to the app to build the connection between the application server and the app. The application sever also has a Sender Auth token to give it authorized access to GCM services. The basic interaction among the three components of GCM is described as follows:

1. The application server creates a Sender ID and Sender Auth token with a GCM server, and registers itself to the GCM server.
2. An app registers itself to the GCM server with its application ID and application server’s Sender ID. The GCM server creates a registration ID and send it to the application server.
3. The application server sends a message to the GCM server; the server enqueues and stores the message, and then delivers it to the app as soon as the device that is running the app is online.

We model the architecture of GCM services in the publish-subscribe style following the definition in [5]. In the architecture, the publisher is the GCM server and its connected application server that produces notification messages and the subscriber is an app which has subscribed the notification messages. The architecture includes several structural components: an app that registers itself to the GCM server and an application server that sends messages to the GCM server for delivering; a GCM server announces

http://www.hipore.com/ijsc/
subscription events and registers each app with its subscribed messages. The publisher and subscriber are bound with the events, and messages are delivered asynchronously by the GCM server. The PrT nets model for modeling the PS architecture of GCM is shown in Fig. 2.

In Fig. 2, each place is a structured data type including two fields: an ID field defines identity information, and a message field defines a structured message that could be a notification message or an action message. Transition Sub defines the subscription behavior, and transition Pub defines the publishing behavior. Place App represents the app that is going to subscribe messages and receive subscribed messages, and transition RecMsg defines the behavior for receiving messages. Place Sender represents the application server that produces messages. Place Event is used to collaborate between transition Sub and Pub to ensure that a subscribed message exists in the publisher. Place Wait is defined for the subscription messages, and transition Notify is defined to register the subscription of messages and send a message of registration ID to the subscriber. Place Auth is defined for the registration ID, and Msg is defined for the registered messages, which stores the matrix that maps the each app to its subscribed messages. Transition Deliver sends a notification message or registration ID to the subscriber. Each transition may include guard conditions to define the firing of the transition in addition to the flow relations that are associated with the transition. Since guard conditions have to be defined with structure details of the messages, they are not defined in Fig. 2 without affection the illustration purposes in this section. Guard conditions will be discussed in detail in the case study section.

We use MISTA to execute the PrT net model for simulating the mobile push notification services and check the interactions among publisher and subscriber. The simulation process is very helpful for developing a correct model. Fig. 3 shows a snapshot screen of the simulation and the corresponding initial marking, which shows an app subscribed an message and will receive a registration ID.

3.2. Mobility

Mobile push notification services can deliver messages to both stationary users and mobile users. A stationary user receives subscribed content with a communication device that has a fixed IP address, and a mobile user may access the service from different locations or during moving. Mobile push notification service providers have to collaborate with networking service providers like ISPs and mobile communication carriers to address mobility issues. There are two scenarios of the mobile usage [14]: (1) A mobile user with an app that subscribed the service moves from one location to the other and the app is connected with different IP addresses before and after the moving, and the app is offline during the moving. As soon as the mobile user is connected to the network in the new location, the app communicates with the networking service provider to locate the app, which is transparent to the mobile push notification services like GCM. (2) During the moving period of time, the app is still online. The connection between the app and the network is supported by the networking providers like mobile phone carriers that resolve the location at run time. Mobile push notification services just deliver messages to registered carriers. Therefore, the subscriber and publisher in the model defined in Fig. 2 have to be connected dynamically at run time to support the two mobile scenarios. We add a dynamically configured connector in the PrT net model to bind a mobile user to its service providers at run time.

As defined in [19], a connector is a PrT net that connects with other PrT nets only through its input and output places. A connector serves as a module for defining the connection policies among nets. We split the PrT net

![Figure 2. A PrT net model of GCM](image)

![Figure 3. An execution snapshot screen of Figure 2](image)
model in Fig. 2 into 2 parts: a publish net and a subscribe net. The two nets are connected with a connector net through its input and output places. The connector net in the PrT net model is shown in Fig. 4, where the dash area is the connector. Transition Route in the connector is used to resolve locations at run time and connect a publisher and its subscribers, and it can be expanded into a PrT net depending on routing policies. Places P10 and P13 are input places for receiving data, and places P11 and P14 are output places for outputting data. Places P7, P8, and P9, and transitions T7, T8, and T9 are places and transitions added for splitting the publisher from the model as a separated net. Transitions T10, T11, and T12 are added for the same purpose for splitting the subscriber from the model as a separated net. The publisher net and the subscriber net are connected with places P10, P11, P13, and P14, and all other parts are same as the model in Fig. 2. Through adding a connector into the publish-subscribe PrT net model, the mobility of subscribers and the dynamic connection between a subscriber and its publisher is appropriately modeled. The location of the moving subscriber is resolved by transition Route, and it is also responsible for connecting subscriber and its publisher according to its routing policies.

3.3. Security

Both GCM and APNS support the communication between mobile users and service providers through secure communication protocols, and the authentication is implemented with secure tokens. In GCM, the Sender ID, application ID, registration ID, sender auth token, and secure communication protocols are used to ensure the security of the communication. Mobile push notification services such as GCM define the security policies and provide security protections. Therefore, we don’t model the security protection policies and process in the model; instead we model the security threats using threat models. A threat model defines whether the service is protected from a security attack. As defined in [22], a threat model is PrT net with one or more transitions that represent a security attack or security vulnerability. Fig. 5 models a possible attack with a threat model for the model defined in Fig. 2. The threat model models a security scenario that an application server (i.e. attacker) forges itself as another valid application server (i.e. victim) to send notification messages to victim apps, which have subscribed the victim server’s messages but not the attacker’s. The attacker gets the Sender ID of the victim server through reverse engineering of a victim app, and steals the sender Auth token from the victim server. The attacker is possible to intercept the sender Auth token if the victim server and the GCM server communicates via a non-secure communication protocol. As soon as the attacker gets the Sender ID and sender Auth token from the victim, it uses them to send messages to GCM server. The GCM server thinks the messages are sent from the victim server and will deliver the message to the victim apps. In Fig. 5, the dash area is a threat model for modeling the potential attack we just discussed. The threat model is connected to the publish-subscribe model via places App and Sender, and transition Pub, and all others parts are still the same. Transition ReverseAttack represents the reverse engineering process for finding the Sender ID; StealToken defines the action for stealing Sender Auth tokens from the victim Sender; MsgAttack is for fabricating forgery messages; places SendID, FakeMsg and Token represent Sender IDs, forgery messages and Sender Auth tokens, respectively. For any attack, we can create a threat model using the same idea.

3.4. Analysis

As soon as the PrT net model of mobile push notification services are built, rigorous analysis of interesting properties of the services can be conducted. The analysis results can help developers to correct and update the model. The PrT models will serves as a design for building systems with the service. In this research, one can conduct simulation and testing of the model during the model development process since PrT nets are executable. Model checking and formal verification also can be conducted on the PrT net model since PrT nets are a formal language. Tool MISTA supports testing and simulation of PrT nets as well as model checking of simple properties. The analysis process of PrT nets using MISTA can be summarized as follows:

1. Compile the model and simulate the execution of the model with initial markings during the model development process to check important execution
sequences and results, look for and correct errors in the model.

2. Define goal states and verify the reachability of the goal states from the giving initial states. States (i.e. markings) in PrT nets are defined by the distribution of tokens in places. Verify the reachability of all transitions from the initial states in the model, and carefully check the transitions that are not reachable. Check for deadlock and terminate states in the model, which is a state under which no transition is fireable. Find reasons that cause the deadlock or termination. Define interested assertions for the model, and verify whether or not each assertion holds at any reachable state [21].

3. Select a test coverage criterion, programming language and test engine for generating tests. MISTA tool can generate adequate tests for the selected criterion. Examples of the criterion include transition coverage, state coverage, goal coverage and others. MISTA supports Java, C#, C++ and other programming languages, and test engine like JUnit and Window Tester. As soon as the model level tests are generated, one can define the mapping between the objects of the PrT net model and the objects of its corresponding implementation, and then convert the model level tests into the implementation level tests using MISTA.

Simulating the execution of a particular sequence of events in the PrT net model is helpful to understand the model and find and correct defects in the model. Fig. 3 is a screenshot of the execution, where the transitions in red are enabled at the current marking. Users can select the next transition to fire or let system to randomly pick one, and the outputs of the execution and the firing sequence of transitions are useful for checking whether a use case scenario is appropriately implemented in the model. Checking the deadlock and termination states and verification of the reachability of goal states, all transitions, and assertions are important to ensure the quality of the model. They provide a more comprehensive and rigorous checking of the correctness of complex models than the simulation does. Test generation is the most important task in the analysis because generating implementation level tests to adequately test the implementation is the main purpose to create it. The purpose of simulation and verification of the model is to create a correct model, and one of the purposes of building a correct model is to drive the generation of high quality of tests for testing the implementation. This type of testing technique called model based testing greatly improves the effectiveness and performance of software testing.

Each security attack or vulnerability is modeled as a threat model to be integrated into the original model. Simulation and verification of the model is helpful to check whether the attack can occur or not and evaluate the possibility of the occurrence. The threat model is also used to produce tests for testing security issues in the model and the corresponding implementation. Testing with threat models is an effective way for security testing because regular models usually model the desired behaviors, which are not useful for generating tests to test undesired behaviors such as security attacks. Threat models specify undesired behaviors like security attacks explicitly, which are used to generate corresponding tests for testing specific security attacks. Using the security attack discussed in section 3.3 as an example, we can check the possibility of the attack using simulation of the threat model. Under the simulation of a given marking, if an attack sequence is fireable, then it means the attack could occur. Fig. 6 is a screenshot of simulating the threat model defined in Fig. 5. Based on the simulation result, one can find that the attack could occur in the model. However, the attack could only happen based on the assumption that the sender auth token could be intercepted.

Figure 6. An execution of the threat model in Fig. 5

In order to generate tests for testing the attack, we define an execution sequence of events (i.e. transitions) that represent an attack, and then generate tests to cover the given sequence using MISTA. For example, an attack sequence of the threat model in Fig. 5 is: ReverseAttack, StealToken, MsgAttack, Pub. The tests generated for covering the sequence is partially shown in Fig. 7. The tests define a set of markings to enable the attack sequence.

Figure 7. An example of tests for testing attacks

http://www.hipore.com/ijsc/
4. CASE STUDY: A MOBILE SEARCHING SYSTEM

In this section, we conduct a case study of a mobile searching system that was implemented with mobile push notification services based on Amazon AWS and Amazon SNS.

4.1 Basic Functions

The mobile searching system we developed is called Hnite, which was originally designed for college students to search local night events such as specials for restaurants and live music events using their mobile devices. The system was extended later for general users to look for local business specials. The system includes two parts: a mobile application that is installed in Android mobile devices and a server program that was implemented with Amazon web services AWS. Amazon SNS and mobile push notifications were implemented in the system with GCM. Local business owners who have registered for Hnite service can enter business data to the server through the webpages linked to Hnite server host in Amazon AWS, and they can update the information at any time. A mobile application user can set searching criteria such as favorites of food, sports, music or grocery. Each favorite may include some sub-categories. For example, food might include sub-categories like casual, exotic, and formal. The criteria also include emotion information such as boring, happy, and normal, and status information like busy and free of the user, and distance such as 5 miles, 10 miles for searching related information within the distance range. Other information like location and weather are sent to the application server automatically when a mobile user moves to a new location and is connected to the network. The push notifications are triggered by local events such as when a local business owner updates the business data, or the context of a mobile application user has changed. For example, a user updated his or her searching criteria, moved to a different place or weather has suddenly changed. The notification messages sent to the application users are fairly brief and informational such as: lunch specials, new pirate music, purple devil tickets for sale. Hnite system limits the length of each message to 100 characters, which is long enough for normal mobile push notification messages. When a user responds to the message via clicking the message, information that is best matched to the notification and user searching criteria is automatically loaded to the mobile device from the server. Then the user can browse or search the loaded information. When a user browses or searches the information, more data might be loaded to the mobile device in the background according to some basic rules such as the search result from the current loaded data is empty, the user expands a top event category but the sub-categories data was not loaded yet or the users browses to the end of the list. Caching data in the mobile device improves searching performance and reliability so that the service is still available when the user is offline. The four screens in Fig. 8 show some basic functions of Hnite: (a) is a list of information pre-loaded in the mobile device according to user’s searching criteria; (b) is a dynamically updated searching result, (c) is a detail view of a selected event, and (d) is a notification message.

4.2 Implementation of Mobile Push Notifications

The server program of Hnite is a regular web server that was implemented in Amazon AWS. Local business owners who registered with Hnite’s service can enter data to the server via the web pages, and mobile application users of Hnite access the server through the mobile application in their Android devices. Mobile users can set up their favorites and search or browse online information in the server or cached data pre-loaded from the server to the device, and the mobile application can send location related data such as location and weather to the server automatically in the background. Fig. 9 shows the overall structure of system Hnite.

![Figure 8. The searching and notification screens](http://www.hipore.com/ijsc/)
authorized. A mobile application installed in a mobile device needs to register itself in GCM and Amazon SNS before it can receive push notifications. When one registers his or her mobile application in GCM, the GCM server generates a registration ID for the device (actually it is for the application, we do not distinguish them as soon as they do not cause confusion). Then the registration ID is sent to Amazon SNS for binding the application and its notification provider [1]. The major event sequence for sending a push notification message in Hnite is described as follows:

1. When a business owner updates information in Hnite server, the server checks the updated information and the favorite setting of each mobile application user to decide whether a notification message is needed to be sent out.
2. If a notification message is needed to be sent to one or more mobile users, Hnite server builds a notification message according to the properties of the updated information and a list of registration IDs of registered users according to their favorite settings, and sends the message and the user list to Amazon SNS.
3. Amazon SNS checks the message and the list of registration IDs, and then sends them to a GCM server on behalf of Hnite server.
4. The GCM server sends the message to all users in the list. It also enqueues and stores the message in case a mobile device is offline. As soon as the mobile device is online, the GCM server sends the message to the device.
5. On the mobile device, the Android system broadcasts the notification to the specified Android application such as Hnite.
6. Users can choose to respond to the notification or ignore it.

Partial data from the server is preloaded or updated into the mobile devices according to favorite setting periodically in the background to improve the searching performance or in case network connection is not available. More data might be automatically loaded when a mobile user browses or searches the preloaded data. Loaded data in the devices can be cleaned or replaced manually or automatically.

4.3 A PrT Nets Model

In this section, we discuss how to model the mobile push notification service in Hnite system using PrT nets. The PrT nets model provides a solid foundation for rigorously analyzing the design and implementation of mobile push notification services. The mobile push notification service includes three parts: a component for connecting Hnite server with Amazon SNS, a component for connecting Amazon SNS and GCM server, and a component for connecting GCM server and mobile devices. Under the guiding of the publish-subscribe model discussed in section 3.1, we model Hnite in publish-subscribe style with more details. Hnite server, Amazon SNS server and GCM server form a publisher, and mobile clients are the subscribers. The registration of mobile clients, the authentication of senders, message generation and delivering are all modelled. Connector discussed in section 3.2 can be easily added to the model to model the mobility of mobile clients, and threat models discussed in section 3.3 can be added to the model for checking security attacks and vulnerabilities.

Fig. 10 is a PrT net model of the mobile push notification service in Hnite. We first consider the case of message generation: as soon as a business owner updated the information in the server, a notification of the change is sent to all registered mobile users. In Fig. 10, place BusinessOwner represents the business owner; three parameters of the place are the ID of the business owner, the message information, and an action. For example, (101, "m", "dir") means a business owner with ID 101 updated information in the server with new information m, and a notification about the change is sent to all registered users. A business owner enters (modeled with transition Enter) the data into the server via a web browser, and the data is added (represented by transition AddData) into Hnite server, which is defined using place Server. The server sends its sender ID to Amazon SNS server and be authorized. Place APIKey has the project key of Hnite server, and it is used to generate senderID (represented by place SenderID) for the server. The server registers (using transition RegSender) the senderID in Amazon SNS server (defined by place SNS). Before a notification message is sent out, the SenderID has to be approved (by transition Approve) and authenticated (by transition Authenticate), and the function of sending message is enabled (defined by transition Enable). When the information in the server is updated, Hnite server generates a notification message (via transition GenMsg) and sends it to SNS. If the message is a topic message and the topic does not exist (handled by transition Compare), a new topic

http://www.hipore.com/ijsc/
We first executed the PrT net model with selected initial markings and checked different execution scenarios until we couldn’t find any error. Then we moved ahead to analyze the model and update the model if any error is found during the analysis.

### 4.4. Analysis

As soon as the PrT net model of the mobile push notification service has been built and successfully compiled, we start to analyze the model for interested properties using tool MISTA. First, the model is executed with random inputs (i.e., initial markings) to help developers understand the model and detect easily found problems. If the simulation result is acceptable, we start the verification of the reachability of goal states, assertions and deadlock states. After that, adequate tests are generated according to selected testing coverage criteria, and finally selected tests are performed to test the model, and the corresponding program level tests generated from the model level tests could be used for testing the implementation of Hnite system. Here...
we discuss the analysis process through analyzing the PrT net model in Fig. 10.

First, execute the PrT net model with some valid initial markings using MISTA. For example, we assign some initial markings for the PrT net model defined in Fig. 10. Considering we need test several functions in the model such as the business owners update the information in Hnite server, and it should trigger a notification message be sent to all registered users or subscribed users. Therefore, we give an initial marking for place BusinessOwner as (101, “m”, “dir”), where 101 is the ID, “m” is the information, and “dir” means the information is sent to all registered users. Obviously, “m” is only a dummy representation of real data using in the system. For similar consideration, we assigned place MobileClients as (1, “r”, “top”), (1, “r”, “sub”), (1, “x”, “reg”), (1, “on”, “swt”) to represent an application user updates his or her context information, subscribes a topic, registers the mobile device, and connects it to the network, respectively. Place APIKey as (1, 12345, “key”), and Topics as (“blk”) mean sending SenderID of Hnite server and creating a blank topic, respectively. We ran the model with the initial markings, tracked the execution sequences, and updated the model based on execution results. Finally, we developed the model in Fig. 10.

Second, verify the reachability of goal states, assertions and deadlock states using MISTA. No deadlock or terminate state was found in the PrT net model in Fig. 10, and all transitions are reachable. We defined and checked several important goals as follows:

1. GOAL (“x”, “m”, “dir”), when a business owner updates information in Hnite server, a mobile client will receive a notification message.

2. GOAL (“x”, “r”, “top”), when a mobile client updates his or her favorite, a notification is pushed to the client. where “x” represents the registration ID of the mobile client, “r” or “m” represents a notification message, and “top” or “dir” means the notification is a subscribed message or a direct message. The analysis found the two goals were all reachable. For checking an undesired goal such as a security attack, one needs to check the negation of the goal to show that the model is safe.

Third, generate adequate tests according to selected test coverage criteria using MISTA. For the PrT net model defined in Fig. 10, we generated model level tests covering all states, all executable paths, and all goal states. We also generated some random tests. Complex execution scenarios can be rigorously tested in model level thanks to the executable capacity of PrT nets. The model level tests are also used for generating program level tests via mapping the model to its corresponding program. The program level tests will be used for testing the implementation of the model. Fig. 11 is a snapshot of the generated tests covering the reachability tree in the PrT net model in Fig. 10, and the number of tests generated for covering the reachability tree is 30849, and they cover total 7712 states in the model. It is not necessary to test all tests for covering reachability tree, so we may manually select some of the tests for the testing. Only 4 tests were needed for adequately testing all transitions, and 1968 tests were generated by MISTA for adequately testing for all states. As soon as tests are available, test engines such as Junit can be used together with MISTA to automate the testing process.

The tests generated in the model level mainly are used for generating program level tests to rigorously test the implementation such as Java programs of the system. The adequate tests generated in the model level are converted to program level tests for testing the implementation to ensure the rigorousness of the test in program level. The tests can be used for testing the programs as soon as the programs have been implemented following the model, but help files are needed to map the differences between the model and its

http://www.hipore.com/ijsc/
implementation. Fig. 12 is a Java test generated from the model level tests partially shown in Fig. 11.

Through rigorously analyzing the PrT net model, and adequately testing its corresponding programs, we have the confidence that the quality of the design and implementation of the mobile push notification is ensured.

5. RELATED WORK

Mobile push notifications are an important feature of mobile applications like those installed in smart phones. It has been implemented in many applications like gaming, financing, sports and emergency systems. However, most mobile push notifications have to implemented based on complex third-party notification services such Apple APNS and Google GCM. It also needs implement complex security authentications and authorization among different parties such as the registration of application servers and mobile clients. It is also a grand challenge topic in mobile push notifications to deal with the mobility issue in the system such as a mobile device may move from one location to another, and a mobile device may join or leave a computing task randomly. For example, a group of players are playing online game, but one of them may pause the playing for a while and then join the game later. Research on mobile push notifications was reported in [14], which discussed the service architecture and different mobile scenarios. However, research on formally modeling and analyzing mobile push notification services is rare.

Formal modeling of software systems provides a solid foundation for designing and analyzing a software system to ensure the quality of software development. A variety of work on modeling and analyzing the mobility in mobile computing systems or services has been reported. The logic agent mobility (LAM) discussed in [19] was modeled using two-layer PrT nets, where the two-layer PrT nets introduced connectors to facilitate the communication between nets. The two-layer PrT nets have the same semantics of PrT nets for each net. Although it is fine to model mobile push notifications using LAM, analyzing LAM model is a grand challenge due to the static definition of the connectors in the two-layer PrT nets, and the complexity of composing several PrT nets in different layers using connectors. We used the connector in this research same as the one in LAM, but PrT nets used in this research is regular PrT nets so that the model can be easily analyzed using MISTA. Reference nets [11] have been used for modeling different types of mobility in [9] through using “nets within nets” style [10]. The “nets within nets” reference nets are natural for modeling mobile push notifications since a mobile computing system can be modelled as a system net and the mobile push notification can be modeled as a token net within the system net. However, analyzing the net needs a new tool that is not available and the semantics of reference nets are different to traditional Petri nets. Therefore, the complexity of formal analysis of reference nets comparing to regular Petri nets is great increased. CPrT nets are PrT nets extended with dynamic channels for synchronous communications and the “nets within nets” style that was introduced in [3]. CPrT nets can model mobile computing systems with concise and easily understandable models. It is able to nicely model the dynamic configuration of software architecture of mobile computing systems. However, an automated analysis tool for CPrT nets does not exist. π-calculus [12] is the first language offering features for formally modeling mobile computing systems via specifying process movement across channels. “π-Calculus is a way of describing and analyzing systems consisting of agents which interacts among each other, and whose configuration or neighborhood is continually changing” [12]. Comparing to π-Calculus, PrT nets are more intuitive for modeling mobile computing systems thanks for its graphic notation. It is also easy to simulate the execution of the net model due to its executable ability. In this paper, PrT nets are used for modeling the mobility in mobile push notifications with a dynamically configured connector.

Many research results on modeling and analyzing software security have been reported. Xu et al. [20] proposed a threat-driven modeling and verification of secure software using Aspect-oriented Petri nets, and they also proposed an approach for automated security test generation with formal models [22]. We adopted the threat model proposed by Xu et al. [22] to model security attacks and vulnerabilities in mobile push notification services, and used MISTA to generate security tests following the same idea discussed in [22].

One of the purposes of formally modeling software systems is to build a foundation for formal analysis. Although formal analysis provides a rigorous way for verifying correctness of software systems, it is infeasible for analyzing large software systems due to the state explosion issue. In [5], He and et. al. reported a method for formally analyzing Petri nets using model checking and other formal proof techniques. Ding and He discussed an approach for model checking a mobile computing system in an extended version of PrT nets [3]. Several other researchers also explored the technique for testing models and designs of software systems [15]. Test generation is the most important task in software testing. The main task of the analysis discussed in this paper is generating adequate tests based on PrT net models. There are mainly four types of test generation: program-based, specification-based, model-based and random test [16]. Program-based test generation falls into two categories: static methods and dynamic methods. Static methods generate test by analyzing the code without running the program, such as symbolic execution [8] and static single assignment form [6]. Dynamic methods produce test using heuristic analysis of runtime behaviors, such as dynamic symbolic execution [17]. Specification-based methods generate abstract test from formal specifications (e.g., in UML [18] and Petri nets [21][23]) and then transform the model level test into implementation
level test. Random test generation selects an arbitrary subset of all possible input values over the input space according to certain probabilistic distribution [4]. The test generation discussed in this paper belongs to specification-based test generation. In order to achieve the rigorous as well as practical modeling and analysis, we model a system using formal language PrT nets and analyze the PrT models using a model-based testing technique that tests models with supplementary of simulation and model checking. The analysis approach has both advantages of informal analysis like testing and formal analysis like model checking. Tests generated from the model not only are used for testing the model, but also are used for generating implementation level tests so that the consistency between a model and its implementation is well ensured, which is absent in most existing approaches [21]. The similarity of the techniques used in modeling and implementation phases is important for sharing analysis results in both phases to improve analysis effectiveness and efficiency.

6. SUMMARY

Mobile push notifications are an important feature of mobile applications. Due to its complexity, formally modeling the mobile push notification is necessary to understand its design and implementation, and rigorously analyzing it is required for building a correct mobile system with mobile push notifications. In this paper, we introduced an approach for modeling and analyzing mobile push notification services in PrT nets and discussed how to model the mobility with a dynamically configured connector and security with threat models. We conducted a case study of modeling the mobile searching system particularly the mobile push notification services using PrT nets, and analyzed the PrT net model using tool MISTA. The case study has shown the effectiveness of the modeling and analysis approach we proposed in this research, and the model we developed is also useful for others who are interested in mobile push notifications. In the future, our research focus is on modeling and analyzing more security attacks and vulnerabilities in the mobile push notification services.

7. ACKNOWLEDGMENT

We thank Zachary Dain for the implementation of partial of the mobile searching system, and Dr. Dianxiang Xu for offering tool MISTA. This research is supported in part by award #CNS-1262933 from the National Science Foundation. Junhua Ding’s research was also partially supported by the guest professorship grant from school of computer sciences at China University of Geosciences.

8. REFERENCES


http://www.hipore.com/ijsc/

Authors

Junhua Ding is an associate professor of computer science with East Carolina University (ECU). He received BS, MS and PhD, all in computer science in 1994, 1997, and 2004, respectively. Prior he joined ECU at 2007, he had worked as a software engineer and project manager with medical companies for 8 years. His research interests are software design and analysis, software testing, Petri nets, and cytometry. He has published 60 peer reviewed conference proceeding and journal papers. He is a member of ACM and IEEE.

Wei Song received the PhD degree in Computer Software and Theory from Nanjing University, China, in 2010. Currently, he is an associate professor at Nanjing University of Science and Technology, China. His main research interests are focused on services computing and software engineering, including software evolution, service composition, business process management, and formal methods like Petri nets. He has published more than 20 research papers in journals and international conferences such as ICWS, SCC, APSCC, etc. He is a member of ACM and a senior member of CCF.

Dongmei Zhang is a professor and chair of department of computer applications at China University of Geosciences (Wuhan). She received BS, MS and PhD. in 1994, 1999 and 2007, respectively. Her research interests are data mining, machine learning, and evolutionary computation.
REFERENCE ARCHITECTURES FOR PRIVACY PRESERVATION IN CLOUD-BASED IoT APPLICATIONS
(EXTENDED VERSION OF 7398 AT MS 2014)

Ivor D. Addo, Sheikh I. Ahamed
Marquette University
Milwaukee WI, USA
{ivor.addo, sheikh.ahamed}@marquette.edu

Stephen S. Yau, Arun Buduru
Arizona State University
Tempe AZ, USA
{yau, arunbalaji.buduru}@asu.edu

Abstract
As the promise of the Internet of Things (IoT) materializes in our everyday lives, we are often challenged with a number of concerns regarding the efficacy of the current data privacy solutions that support the pervasive components at play in IoT. The privacy and security concerns surrounding IoT applications often manifests themselves as a threat to end-user adoption and negatively impacts trust among end-users. In this paper, we present a reference software architecture for building cloud-enabled IoT applications in support of collaborative pervasive systems aimed at achieving trustworthiness among end-users in IoT scenarios. We describe a case study that leverages this reference architecture to protect sensitive user data in IoT application implementation. We then evaluate the response data from our end-user survey. In addition we present a Secure, Private and Trustworthy protocol (named SPTP) that was prototyped for addressing critical security, privacy and trust concerns surrounding mobile, pervasive and cloud services in Collective Intelligence (CI) scenarios. We present our evaluation criteria for the proposed protocol, our results and future work.

Keywords: Software Reference Architecture, Cloud-Enabled Service Privacy and Security, Internet of Things, Collective Intelligence

1. INTRODUCTION

Trends pertaining to the use of the Internet of Things (IoT) to collaboratively solve complex problems in healthcare monitoring, online web site advertising, and smart home implementation scenarios, among others, are expected to continue to grow rapidly. Nonetheless, one of the most critical deterrents to mainstream user adoption of IoT systems is a looming distaste for how data privacy is enforced among the various collaborative systems at play in IoT applications. Beyond the privacy and security concerns encircling IoT systems, it is becoming more and more inevitable for pervasive collaborative devices to leverage web services for data sharing and communication to backend storage systems. With the advent of cloud computing, it is not uncommon for the mobile services that these pervasive devices communicate with, to be hosted in the cloud. Consequently, with the imminent domain-specific privacy and security concerns for cloud computing, IoT, pervasive systems and web services, it is important to establish a reference architecture that provides a holistic solution for implementing cloud-enabled applications and service interactions in IoT scenarios in a fashion that improves the overarching goal of attaining end-user trust and, in turn, improve user adoption of IoT applications (Apps).

We consider some of the key spheres of significance in arriving at a reference architecture that is aimed at achieving trustworthiness among end-users in IoT applications, as being reminiscent of the implementation of security and privacy in:

- The IoT application, holistically
- Ubiquitous computing systems in the solution
- Participating Cloud computing systems
- In the Service-Oriented Architecture (SOA) layer

As the adoption race for the aforementioned innovative computing paradigms continue to mount, researchers have uncovered some of the critical concerns and proposed solutions to various facets of the all-inclusive problem that we seek to address.

In the face of recent attempts at establishing security and privacy frameworks to support trust management in pervasive systems, a comprehensive model that fosters trust among the target users of these emerging technologies is yet to achieve mainstream adoption [21]. Among others, Itani et al. [1] described a set of security protocols for safeguarding privacy and compliance of end-user data in cloud computing scenarios. Yau et al. [2] looked at performance and security tradeoffs in SOA, while Ponemon Institute [3] shared an insightful research report detailing proactive steps to protect sensitive information in the cloud.

A number of previous studies have proposed privacy and security frameworks for protecting data in specific domains ranging from mobile health monitoring [30], self-improving smart spaces [19], location privacy in mobile computing [31], privacy protection in web services [32], privacy enhancement in platform-as-a-service cloud computing scenarios [10], and
more. We challenge the trend by proposing a holistic view to the problem with a focus on protecting the security and privacy of end-user data while fostering trust in new and old technology solutions that are willing to subscribe to these standards.

To throw some light on how various facets of security and privacy implementations can be considered in a holistic reference architecture for realizing end-user trust in IoT applications, we consider a case study where a cloud-hosted Movie Recommendation Service (achieved by leveraging both YouTube and Facebook Services) draws end-user media content viewing behavior and preferences in a multi-member family household from a TV-mounted Microsoft Kinect sensor device, a Smart TV and an Apple iPAD tablet device collaboratively. The Microsoft Kinect Sensor is used to collect information about individual user preferences for movies in a multi-member family household. The movie recommendations service determines recommendations for movies that each household member or a combination of household members might enjoy based on the previous viewing history of each individual household member (captured by the Kinect Sensor and the iPad tablet device) in conjunction with the content viewing preferences of a given household member’s influential Facebook friends. The ubiquitous Kinect Sensor device is used to identify specific users assembled in front of the family TV. Information gleaned by the Kinect sensor is transmitted through a mobile service to the cloud storage location. The Smart TV records the time of day that a movie was watched by a given household and saves it to the cloud storage through a cloud-hosted mobile service.

Our Contributions

Our contributions take the form of a conceptual Reference Architecture for building a security, privacy, and trust management protocol (SPTP) that is capable of protecting private data at the time of disclosure or collection, in-transit, at-rest and for the life of a private data element even when it crosses the boundaries of the original system to be consumed by another system. In addition, we propose a logical Reference Architecture for building cloud-enabled IoT applications.

We also propose a Secure, Private and Trustworthy Protocol (SPTP) with an associated seal that will be readily recognizable by end-users in various online and ubiquitous computing settings. The standard seal is to be used in all systems (including cloud services, mobile devices and applications, sensors, gadgets, web sites, and more) that wish to identify themselves as being secure, private and trustworthy to end-users and other entities.

We describe our motivation for this study in section 2 and showcase a number of relevant application scenarios in section 3. Our resolute contributions are clarified in section 4 through the:

- Discussion of the implementation details for our proposed SPTP protocol.
- Depiction of our proposed reference architecture for supporting privacy and security in IoT systems. We consider these design artifacts as one of our key contributions.

Implementation details for two case studies are also described in section 5. A comparison of our approach with that of previous research work is presented in section 6. In section 7, we describe our evaluation and findings regarding the perception of trust among end-users in our IoT system prototype which seeks to implement our proposed reference architecture in the case study. In the light of our findings, we share our conclusions in section 8.

2. MOTIVATION

This study advances our current investigations concerning the application of cloud services and collaborative ubiquitous devices in mobile health (mHealth) intervention scenarios. In most instances, sensitive health-related data is collected and stored in a cloud-hosted solution. The reference architecture presented in this paper provides a framework for ensuring that privacy and security take center-stage throughout the application development lifecycle in the pursuit of maximizing the promise of IoT, Ubiquitous computing and Cloud computing archetypes.

In addition, we seek to share a blueprint for developing software architecture that supports privacy and security in IoT solution designs. We envision that our template solution for the IoT domain can be adopted, refined and extended by software designers, engineers, and architects who seek to preserve trust across IoT implementations. By promoting the adoption of this reference architecture, we hope to institute a foundation for ensuring consistency in addressing privacy and security concerns among current and future IoT implementations. Previous studies [4] suggest that establishing a reference software architecture promotes re-use, reduces maintenance costs, and serves as a benchmark for software governance. We expect this reference architecture to serve IoT-related solution designers in the same capacity.

3. SCENARIO ANALYSIS AND PROBLEMS

We strongly believe that the scenarios presented in this literature constitute examples of how SPTP can be used to drive technology adoption in several scenarios only limited to the imagination of solution providers and end-users. The two scenarios considered for the SPTP prototype include geographical location monitoring to support online content recommendations, and a ubiquitous health application showcasing elderly care monitoring in a smart environment.

A. Movie Recommendations Engine

We consider a scenario where an Online Social Networking (OSN) solution provider (in this case, Facebook.com) allows end-users to disclose personal information on a user profile page where the end-user’s movie preferences are captured and
shared with friends on the OSN web site. We then establish a third-party online video streaming and recommendations solution provider that only exposes its Mobile Application (App) and web site to end-users who have a valid Facebook credential. Facebook authentication is achieved through OAuth over HTTPS. This movie recommendations App is capable of tapping into a logged-in user’s social graph through the Facebook Graph API [5]. Imagine the movie recommendations solution provider (named Zoei) is a start-up company facing pressures to generate revenue to sustain its services.

The simulated movie recommendations App, in turn, surfaces and ranks movie recommendations based on the end-user’s current geographical location – gleaned from the end-user’s mobile device GPS sensor. The recommendations engine also pervasively retrieves additional user profile data from a logged-in Facebook user which has nothing to do with the perceived “good intent” for retrieving movie title “likes” and recent location data from the end-user’s Facebook friends in support of the movie recommendation ranking feature. In this case, the rationale for this “questionable” pervasively mined data is to sell the retrieved data to other advertising affiliates who have an interest in using some of the end-user profile data to enhance their online Ad targeting and general market segmentation efforts. In addition, this App communicates with a cloud-hosted web service to store the mined data in a public Cloud.

**Key Privacy Challenges:**

- **False Sense of Trust by Affiliation:** The end-user might wrongly perceive the entire solution to be secure and trustworthy because the end-user’s current geographical location – gleaned from the end-user’s mobile device GPS sensor.
- **Access to Private Data Storage:** Without access to the innards of Zoei, the end-user is blind-sided by Zoei’s clandestine collection, storage and transmission of personal identifiable information (PII) and other private data elements.
- **Ongoing Certification and Reputation Status:** The end-user has no way of determining Zoei’s current reputation for privacy and security when he/she accesses the App on the tablet device.
- **Data Privacy Access Control:** Once the profile data crosses the system boundaries of the OSN, there’s no guarantee of access control protection.
- **Privacy in Public Cloud Storage:** There’s no guarantee that PII or private data stored on Zoei cannot be leaked to other solution providers sharing the public cloud’s resources.
- **Web Cookies:** Control over data that is made accessible to third-party web sites is desired.

**B. Multimodal Health Monitoring in Elderly Care**

Another application scenario that we considered include a futuristic setting in which an elderly person is able to interact with a multimodal health monitoring system that is capable of tracking the participant’s physical activities, habits and private health information.

The user behavior data collected about the elderly person is posted through a web service hosted on a PaaS cloud. The scope of activity monitoring facilitated by the Kinect sensor includes the participant’s time spent walking around the house, indoor localization, screen capture and timestamps for medicine consumption, time spent watching TV, interactions with the Humanoid companion, etc. [12].

**Key Privacy Challenges:**

Some unique concerns in this scenario include:

- **Informed Consent:** Participants will prefer to be notified when both sensitive and non-sensitive data is collected about them in the smart environment [24].
- **Ongoing Reputation Access:** After the participant reviews and consents to the program, ongoing access to the privacy reputation of the solution at any point in time and opt-out options is desirable.
- **Data Access:** In some cases, participants will be interested in having control over the data that is collected and how the data is used.
- **Hardware Vendor issues:** A standard that is adhered to by all sensors in the multimodal environment will be ideal.
- **Compliance:** In a ubiquitous health (uHealth) environment, HIPAA compliance becomes a standard of interest, particularly in the US.

**4. PROPOSED MODELS**

In the ensuing section, we describe the characteristics of our proposed conceptual reference model, a security, privacy and trust protocol and also share a number of illustrations of the reference architecture model.

**4.1 Key Characteristics of the Conceptual Reference Architecture**

The reference software architecture for a given domain seeks to define the underlying components of the domain and their associated relationships [4]. The software architecture of a given implementation in the domain then becomes an instance of the domain reference architecture.

The Internet of Things can be described as an amalgamation of multiple heterogeneous digital devices, people, and services working collaboratively to solve a problem across technology boundaries, with the ability to seamlessly interact and share data about themselves and their environment [6]. In deriving our reference architecture, we outline the major components of modern IoT systems that can benefit from preserving security and data privacy by considering some of the fundamental components of IoT systems in the following applicable scenarios:

- **SCENARIO 1: A home automation monitoring service** that is capable of observing the usage of electricity in a given household and seamlessly
regulates the home’s consumption of electricity by learning the preferences of the household in comparison with optimal power consumption best practices of other neighboring households of similar size

- **SCENARIO 2**: An Online Social Networking (OSN) service that is integrated with a pervasive eyewear device, like Google Glass, for capturing pictures and recording videos of interesting moments. The service is assumed to be useful for storing the recorded photos and videos in cloud storage while enabling the end-user to share the stored media with other friends in the OSN as well as public access.

- **SCENARIO 3**: A movie recommendation service that leverages previous media content viewing patterns and the preferences of influential people in a particular household user’s circle of OSN friends to recommend future movies that will be of interest to the user. This scenario is employed in our case study implementation. A contextual view of the IoT-based movie recommendations scenario is illustrated below in Figure 1.

### Logical View of Interactions in the IoT System

![Fig. 1. Logical View of Interactions in the IoT system](image)

In line with the scenario presented above, some of the major components that have their own facets for privacy and security concerns include:

- **End-User Preferences** for Security, Privacy and Trust
- **Cloud Computing**: in the form of a cloud-hosted web or mobile service and cloud-based data storage
- **Ubiquitous Computing**: represented by the Kinect Sensor, Tablet device and a Smart TV
- **Service Oriented Architecture (SOA)**: in the form of the Facebook Graph API (web service) used for inferring the preferences of influential friends in a given household member’s OSN circle as well as the YouTube API (web service) for streaming a movie
- **Network communication** across wireless networks for transmitting and receiving data between the ubiquitous devices in the Smart Environment and the external cloud service environment

Conceptually, most of the major facets of a generalized IoT implementation are likely to include the fundamental components captured in Figure 2.

End-users have preferences for security, privacy and trust that must be collected and adhered to at all facets of the solution. There is typically an optional user interface and a physical sensor or ubiquitous device for data collection. These physical data communication components communicate to external systems through a network communication layer using a communication protocol to a web or mobile service of some nature. These web services, in turn, persist streams of data to a backend storage device. Typically, the backend service engine and storage is hosted in the cloud.

### 4.2 The SPTP Protocol

In arriving at a solution to user privacy standardization gaps in uHealth and online advertising scenarios, we propose a protocol that is capable of tagging private data with an access control list (ACL) that can be defined and managed by end-users across multiple platforms. In addition, we propose that the protocol is applied to data in-transit and at-rest. The protocol should be able to retrieve and validate the privacy policy of a given web page or ubiquitous system against the privacy ACL defined by the owner of the private data. The data owner should also be able to gain access to his or her data and make a decision to opt-out when need be. The end-user should also be able to observe the current privacy reputation score of the subscribers to the protocol.

In addition, the protocol is designed to be administered and enforced by a third-party regulating body to create an unbiased regulation of privacy standards and policies aimed at protecting the end-user. Most importantly, the protocol implementation must ensure ease of implementation to avoid the previous plight of P3P. We prefer to look at a holistic standard protocol that can be used to regulate user data privacy and security across ubiquitous and traditional web solutions to ensure consistencies in expectations across various mediums, domains and scenarios. The conceptual architecture of the SPTP protocol is described in the context of a case study in section VII.
4.3 The Reference Architecture
In creating a generalized reference architecture for improving security and privacy concerns in IoT systems we illustrate the various layers and characteristics discussed in section III along with some of the best practices employed in security and privacy implementations in Figure 4.

In most cases, end-users are likely to accept an IoT solution that is managed or hosted on a trusted cloud provider system. We propose the use of a governance body for ongoing certification and regulation of standards pertaining to the all-encompassing extent of a typical IoT implementation.

The ensuing literature describes some of the key concerns inherent in each layer of the IoT reference architecture.

4.3.1 Privacy in the Ubiquitous Sensors and Devices in the Smart Environment
In considering the security and privacy concerns of IoT applications, it is important to hone in on some of the security and privacy challenges pertaining to pervasive devices and sensors that are often working ubiquitously to collect and exchange data in the environment. From a security and privacy perspective, some of the key requirements that can be addressed at this layer of the IoT application include [6]:

- **User identification and validation** to control access and enforce permissions and authorization levels for various components of the system
- **Tamper resistance** of the physical and logical device. Because IoT devices are typically unattended, physical attack vulnerabilities are critical.
- **Content security** - through digital rights management (DRM) of content used in the system
- **Data privacy** to protect sensitive user data.
- **Data communications and storage security** through protective measures for both data in-transit and data at-rest
- **Secure network communications** to ensure that network communications between ubiquitous devices and external services are only authorized through secure connection channels (for example, the wireless communication in the smart home environment of our case study can only be transmitted through the user’s designated “home” wireless router, by default. Eavesdropping vulnerabilities in the wireless network must be curbed.
- **Privacy in ubiquitous computing** comes to play because the way in which the device or sensor collects data about the end-users might conflict with the user’s privacy preferences for a particular scenario. For example, while a user might be open to having the Kinect sensor perform user identification to support the IoT system, he or she might not be open to having the Kinect record certain conversations in the smart home living room. These privacy barriers and preferences must be preserved in order to instill end-user trust in the system
INTERNET OF THINGS (IoT)
CONCEPTUAL REFERENCE ARCHITECTURE: CONTEXT VIEW

- **End-Users**
  - **Personas**
  - **Smart Environment**
    - **User Interface**
    - **User Preferences**
  - **External Environment**
    - **Third Party Services**
  - **Capabilities**
    - **Security**
    - **Privacy**
    - **Trust**
  - **Network Communications**
    - **Ubiquitous Sensors & Devices**
    - **Cross-Cutting Governance**
      - **Enforce End-User Preferences**
      - **Standard Policies & Procedures**
        - **Device Certification**
        - **Cloud Provider Regulation**
        - **IoT App Certification**
        - **Establish Trust Levels**
        - **Standard Protocols**
  - **Cloud Environment**
    - **Services Layer**
      - **SAAS**
      - **PAAS**
      - **IAAS**
      - **Virtualization & Abstraction Layer**
        - **Physical Resource Layer: Facility + Hardware**
  - **INTERNET OF THINGS (IoT)**

That notwithstanding, some of the key security features that can be incorporated in pervasive device architecture (that is considering both hardware and software requirements) include [6]:

- Lightweight cryptography approaches to support the low power, memory and processing power constraints in most pervasive sensors and actuators
- Security measures to protect the physical device
- An interface for determining and controlling the privacy preferences of the end-user. This might include a visual cue (for example, a green light) when the device is in recording mode. The ability to stop or pause the recording mode easily becomes critical as well
- Standard security protocols for direct device-to-device communication and device-to-service communication. For example, the Smart TV in our case study might need to send the user data securely to the cloud service in our case study through a secure socket layer (SSL) protocol over HTTPS communication.
- Secure on-device storage including RAM, Flash or ROM storage

- Secure operating system to protect data during runtime execution

In everyday human interactions, trust is often demonstrated when a user is able to confide in a close friend by sharing private or confidential information with the friend knowing that the friend will respect the display of trust by protecting the information from leaking to other unauthorized parties. Consequently, privacy goes hand-in-hand with trust. Privacy is sometimes defined as a critical human right “to be left alone” or an elementary need for an end-user to establish and maintain a private barrier to protect his or her information [7]. In devising a privacy solution at this level, Solove’s [8] taxonomy for addressing privacy might be considered during information collection, information processing, information dissemination, and in curbing invasion.

In smart pervasive environments, the information collection process is often invisible to the end-user by virtue of the built-in ubiquity of the solution. This often raises concerns among user privacy experts. In most cases, the end-user does not have access to the information that has been collected about his or her activities. Equally, the end-user often has minimal visibility and control over the processing and dissemination of the data collected in the smart environment. However, IoT promotes a lot of data sharing among multiple devices. It is
more critical for devices in an IoT setting to conform to standard privacy protocols that provide end-users with a level of comfort and trustworthiness that their private data will be protected. These issues have to be considered in deriving an effective architecture for hardware and software privacy.

4.3.2 Privacy in the Cloud Computing Layer

Cloud computing can be defined as “a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction” [9]. In a typical cloud system model, there is a [1] Cloud provider – who exposes cloud services, and a Cloud consumer – who consumes these services. In our reference architecture, the IoT App Provider is also considered as a Cloud consumer.

Vulnerabilities in cloud systems can be categorized as [13] being related to Cloud Multi-tenancy, Elasticity, Availability of information (SLA), Information Integrity and Privacy, Secure Information Management, and Cloud Secure Federation. It is important the IoT provider considers these vulnerabilities in arriving at a secure solution.

Nonetheless, vulnerabilities in cloud solutions can differ for a given cloud deployment model. Some of the cloud deployment models in use today include [14][15]:

- Private Cloud
- Community Cloud
- Public Cloud
- Hybrid Cloud
- Virtual Private Cloud

Beyond the considerations for each cloud deployment model, there are unique security and privacy issues in each delivery model. We consider the following layers in an IoT solution that makes use of a public cloud deployment solution:

- Services Layer - which includes:
  - Software Applications
  - Data management systems
  - Operating Systems
- Server Virtualization layer
- Physical Hardware layer - which includes:
  - Physical hardware
  - Network communication infrastructure

Some of the popular cloud delivery models [13][14] include:

- Infrastructure-as-a-Service (IaaS): Where the cloud provider offers storage and computing services on-demand. The cloud customer manages the virtual machines (VMs) and other associated infrastructure components hosted in the cloud – including data storage, operating system and applications hosted on the VMs. Resultantly, security and privacy concerns at the application and operating system level are managed by the cloud customer. The cloud provider will handle security and privacy concerns at the datacenter hosting level including the virtualization and physical hardware layers.

- Platform-as-a-Service (PaaS): In this delivery model, the cloud provider exposes a set of services and application protocol interfaces (APIs) for developers to host web sites and services without having to deal with the scalability issues of the application as the solution usage grows. The cloud customer will have minimal control over the security practices used at the operating system-level.

- Software-as-a-Service (SaaS): In this scenario, the cloud provider exposes specific applications to consumers for use with a multi-tenancy approach that might use a subscription-based pay-per-use model. The Cloud Consumer will have even less control over the privacy and security implementation in this cloud-hosted solution since the entire service layer is managed by the cloud solution provider.

For a cloud consumer in an IoT setting, the trust model detailed in Itani and Kayssi’s findings [1] can prove to be useful:

- Full Trust – where insensitive data is safely transmitted, stored, and processed without encryption on the cloud service
- Compliance-based Trust – where sensitive consumer data needs to be encrypted and sometimes anonymized in support of legal compliance regulations. An example might be compliance requirements for the Health Insurance Portability and Accountability Act (HIPAA) in mHealth scenarios.
- No Trust – where highly-sensitive customer data must be concealed from the cloud provider.

4.3.3 Privacy in the IoT Apps and Service Layer

Security issues in integrating mobile agents and devices with services can be categorized as [18]: Confidentiality, Authentication, Authorization, Integrity, Nonrepudiation, Privacy, and Availability. In most typical IoT App scenarios multiple service endpoints were employed in the solution.
We believe that a third-party governance institution that can address the need for third-party assessment, regulations at all layers of IoT systems, transparency, policies, standards and ongoing status certification will be greatly beneficial to improving the trustworthiness of IoT applications.

B. Health Information Security and Privacy Compliance

In spite of the adoption and improvement of health information privacy protection in the US through the Health Insurance Portability and Accountability Act (HIPAA) Privacy Rule [28], there are ongoing concerns about data privacy associated with electronic health solutions.

While it might seem important that ubiquitous and cloud technology providers comply with HIPAA and PCI compliance requirements, these organizations only regulate healthcare and Ecommerce applications. This leaves gaps for non-compliant vendors to remain susceptible to other vulnerabilities that can cost end-users notable security and privacy breaches.

To present an overarching solution that will protect end-users from the security and privacy vulnerabilities inherent in pervasive and cloud systems, we strongly believe that third-party organizations may need to step in and develop compliance requirements that can be used to certify and regulate cloud providers and cater to the various cloud service delivery and deployment models discussed above. SPPC can then be employed to communicate and verify compliance to the standard security and privacy policies.

5. CASE STUDIES

We share two case studies that we used to validate the feasibility and validity of the proposed models described earlier.

5.1 Case Study for The Reference Architecture: A Movie Suggestions IoT Application

We implemented a prototype solution for a personalized movie recommendation IoT Application that takes into account our proposed reference architecture in order to derive the IoT Application’s solution architecture for governing security and privacy concerns at various layers of the solution.

Personas or Actors identified in the reference model include:

- End-users of the IoT solution
- Device providers or vendors who build pervasive sensors and devices that are used in IoT systems. For the Kinect sensor, the device vendor is Microsoft.
- IoT Application Provider and Cloud Consumer
- Cloud Provider who exposes cloud services that are used by IoT Providers. For our case study, the Cloud Provider was Microsoft’s Azure Cloud. The third-party OSN provider in our case study was Facebook’s Graph API while movie trailers surfaced in the App were served through Google’s YouTube API.
- Our pseudo third-party Security and Privacy governance provider is expected to take on the regulation and certification of various components in

http://hipore.com/ijsc
the IoT context including the cloud service and storage, external services like the Facebook API, and the IoT application itself. The provider also ensures adherence to standard protocols like OAuth, SSL, etc. across the communication channels.

The external services employed include the Facebook Graph API and the YouTube API. To access the household member’s Facebook friend information the privacy of the user’s data must be considered in the external system. Access to the end-user’s Facebook information is controlled by Facebook’s implementation of a secure OAuth 2.0 over HTTPS protocol for user authentication. In addition, Facebook allows the household user to authorized specific access to the IoT system, after the user’s credentials have been successfully verified. An access token is required by both the YouTube API and the Facebook API for the IoT endpoint to exchange data with the third party services.

For our case study, the IoT application’s user interface takes the form of a movie recommendations engine accessible through both an iPad device and a Smart TV as a web-based application. The user interface for our movie recommendations IoT App is represented in Figure 6 below.

![Figure 6: Movie recommendations IoT App](http://hipore.com/ijsc)

The network communications component in the smart environment, in our case study, represents a wireless router for secure communications between the smart devices and cloud-hosted IoT web service. The ubiquitous computing device in our case study is represented by the Kinect for Windows sensor connected to the Smart TV.

The movie suggestions that are provided in the application are ranked based on the social influence of a particular household user’s OSN friends as well as the previous movie preferences of the OSN friends (captured as Facebook “Likes”). In addition, movies that are similar to media content that was previously identified by the Kinect sensor as part of the authenticated user’s viewing history are factored into ranking the recommended movies for that particular user. While the benefit of this scenario seems to outweigh the need for the Kinect sensor to monitor the user’s viewing behavior, there can be a number of privacy and security concerns that might deter end-user adoption.

The cloud system model used in our case study analysis is synonymous with that of a public cloud where the cloud provider manages and exposes storage and computing services through a geographically dispersed and on-demand scalable cloud infrastructure. We (as the cloud consumer) then utilize cloud storage to host data gleaned from the IoT device sensors. The web services employed in the IoT scenario also operate as cloud services hosted on the PaaS cloud infrastructure.

**Improving Privacy Concerns in this Case Study:**

Some of the concerns considered in enforcing privacy and security best practices at various layers of the reference architecture include:

- **Informed Consent:** We learned that end-users preferred to be notified when the Kinect sensor is collecting both sensitive and non-sensitive data in the smart environment. A visual cue by the form of a blinking green light indicator on the device while it is in recording mode proves to be useful.
- **Control over Privacy Settings:** The parent’s in the household may not want their children to have access to uncensored content, so the parent might want control over media content suggestions that are surfaced to the children in the household. Also, the parent might want to limit how much data is stored by the IoT App (for example, exclude geo-location information from data collection).
- **Vendor Regulation:** A third-party regulation body could be employed to monitor and expose gaps in the system based on the end-user’s pre-defined privacy and security preferences. Ongoing risk assessments on behalf of the end-users could prove to be useful.
- **Access to User Data and Opt-Out:** In some cases, participants prefer control over the data that is collected about them. End-users are also interested in how their data is used and seek to reserve the ability to opt-out and delete their data at will.
- **Ongoing Reputation Access:** Beyond the participant’s initial consent to allowing the IoT App access to his or her Facebook (OSN) data, it will be useful if the participant can access the privacy of the IoT’s solution itself at any point in time and opt-out without any loom of lock-in.
- **User Identification and Authentication:** If the identity management system throughout the IoT implementation is not accurate, uncensored content that might be appropriate for the parent but
inappropriate for the child might be surfaced mistakenly to the child.

- **Physical Security and Wireless Networks:** Prevention of eavesdropping in the wireless network as well as measures to enforce security of the physical objects in the environment proves to be critical.

### 5.2 Case Study For The Sptp Protocol: Elderly Care Monitoring

To test the efficacy of our proposed protocol, our simulation introduced a futuristic smart-environment where an elderly person’s physical activity will be monitored by a Microsoft Kinect for Windows Sensor. The Kinect sensor is mounted on a television in the elderly person’s living room. In addition, the subject is able to periodically interact with a NAO Humanoid robot as a personal companion. Private data collected by both the Kinect Sensor and the Humanoid Robot is transmitted and persisted in an SPTP tagged form through a Windows Azure Cloud Service (PaaS) to a storage account. The simulation seeks to demonstrate the use of several heterogeneous systems in a collective intelligence-inspired solution for activity monitoring [11]. The Kinect sensor is able to transmit data to the cloud storage service through a wireless (WiFi) connection as depicted in Figure 2. It also displays a consistent visual cue when in recording mode.

In this case, the user’s privacy preference ACL is applied to each recorded data segment in the form of a tag that can only be decrypted using the user’s privacy identifier. An informed consent form is presented to the elderly person describing the scope of monitoring and his rights to the data. A digitally-signed online form indicates the type of activity that will be monitored and stored. The user’s preferences are captured as an ACL tag that is subsequently applied to future recordings. The elderly person is also able to review information collected in the smart environment (through a web portal) to ensure that it conforms to his or her predetermined privacy preferences.

When the user decides to revoke a portion or all of the data that was previously collected about him or her, there is an option available through the web site to facilitate this. The user’s preference is then subsequently honored in all third-party systems that have previously consumed the private tagged data within a period of time – assuming those services also subscribe to the SPTP protocol.

### 6. RELATED WORKS

While several security and privacy frameworks and reference architectures exists for various domains, we are not aware of a generic reference architecture that caters to modern IoT App scenarios. NIST [22] proposed a reference architecture for Cloud Computing and Itani et al. [1] have explored a reference framework for privacy in cloud computing scenarios. Several studies have focused on legal compliance and the technical implementation of trust models, frameworks and protocols. Additionally, Langheinrich [23] addressed key privacy concerns in ubiquitous systems.

Recent studies in this area can be categorized as either high-level frameworks (with a focus on legal compliance and risk assessments) or low-level frameworks (with a focus on technical implementation of access controls to data). Neither of these approaches offers a panacea.

#### 6.1 P3p – Privacy Preferences Platform

Arguably, the most popular privacy protocol that is synonymous with our approach is the Platform for Privacy Preferences (P3P) project [25] [26]. P3P turned out to be difficult to implement and further work on the protocol has been suspended. We propose SPTP as a generic security, privacy and trust protocol that will transcend web and other ubiquitous computing scenarios, whereas the domain of focus for P3P was web-based solutions.

However, the mission of SPTP is consistent with the goals of P3P in enabling web sites to express their privacy practices in a standard form that can be verified by user agents [26] and make it easier for end-users to recognize the level of privacy compliance of a given solution without having to read the full privacy policy [25]. One of the limitations of P3P is that, while it facilitates better communication about privacy policies it does not act as an enforcement mechanism for privacy.

---

http://hipore.com/ijsc
6.2 Other Privacy Frameworks

Alpcan et al. proposed the use of a “novel probabilistic diffusion scheme for detecting anomalies possibly indicating malware which is based on device usage patterns” [27]. Behl and Behl [26] recommend solutions to protect end-users from the potential privacy and security vulnerabilities of cloud systems. From a best practices perspective, Yale University [14] proposed some useful security and privacy approaches to support HIPAA compliance.

7. DATA ANALYSIS

We describe our evaluation of the two reference architectures as follows.

7.1 Reference Architecture Evaluation

To test the hypothesis that our reference architecture is feasible for most modern IoT implementations, we reviewed the implementation of security and privacy solutions aimed at achieving trust in the three scenarios (among others) described in section III and compared the inherent layers in the scenario to that of our proposed conceptual reference architecture.

In addition, we built a prototype of the case study described in this paper and shared the way in which various security and privacy concerns are addressed in our work-in-progress solution with a few technologically savvy end-users. We collected feedback from these end-users through a survey to gauge their comfort level with the implementation of security and privacy solutions at various layers of the conceptual model.

Of the 18 distributed surveys, we received 14 usable responses with questions centered on evaluating the importance of the various facets in the reference architecture and how it affects their overall level of trust in the IoT system:

- Trust in the underlying Cloud data storage system
- Trust in the ubiquitous devices and user interface

Participants (end users) were asked to indicate their rating with a scale of 1 through 3 (where 3 means the characteristic is Important, 2 represents Indifferent, and 1 denotes Not Important). Our analysis of the survey response indicates that most savvy respondents (92.8%) were most comfortable with the IoT system when the immediately visible device or sensor exhibited security and privacy best practices.

7.2 Sptp Evaluation

To evaluate the SPTP protocol, we plan on employing a number of approaches including:

- User survey on the perceived benefits of SPTP
- Performance measurement access control
- Analysis of the SPTP protocol and its impact on Ubiquitous system adoption and trust management.

Fig. 9. Savvy End-User Response
7.2.1 User Survey on the perceived benefits of SPTP
To test the hypothesis that SPTP is likely to inspire trust in the adoption of ubiquitous systems among adults, we conducted a preliminary survey to qualify users for our long-term study after demonstrating the proposed use of the protocol in a simulated smart environment for elderly care monitoring and interaction with a humanoid robot.

Of the 20 distributed surveys, we received 14 usable responses. The questions asked in the survey were categorized as Consent, Cues, Access, and Reputation. The survey respondents were also categorized as technologically savvy (9 participants) and non-savvy end-users (5 participants). The age range of the respondents fell between 25 and 53 years old. Participants were asked to indicate their rating with a scale of 1 through 3 (where 3 means the characteristic is Important, 2 represents Indifferent, and 1 denotes Not Important). The results of the user study are illustrated in Figures 3 and 4.

Our analysis of the survey response indicates that most savvy respondents (66.7%) are concerned with having some form of verbal or visual cue present when information that they perceive to be private is recorded and transmitted.

In general, users concluded that having a dedicated third-party service that regulates and certifies security and privacy in these environments could contribute to their level of trust in the system. Most of the non-savvy users expressed strong interests in having the robot instill a sense of connectedness and show unconditional care as a key driver for trustworthiness of adopting the simulated smart-environment.

7.2.2 Performance measurement of access control lists
We plan on evaluating the performance impact of adding the proposed ACL tags to data transmission and storage. Accordingly, we hope to minimize any additional overhead that might be caused by pursuing this approach.

7.2.3 Analysis of SPTP’s Impact on Trust
Once the prototype is fully developed, we plan on conducting a survey with a wider audience to fully expound the promise of SPTP in ubiquitous and online systems. Considering the extent of the project, there is still a significant amount of work to be done to fully test our hypothesis regarding the impact of SPTP on trust management.

8. CONCLUSIONS
With the outburst of cloud services and the advent of pervasive and context-aware services, it is increasingly necessary to ensure that sensitive data is not compromised.

8.1 Sptp Protocol
As the results of our survey indicates, the savvy users of these technologies will have more trust and confidence in cloud-enabled ubiquitous solutions if they can garner some form of assurance that a third-party privacy protocol that enforces compliance standards has certified the application.

Similar to the Data Security Standards (DSS) compliance requirements that often governs the Payment Card Industry (PCI) we envision that it will be useful for third-party entities to adopt our proposed protocol or a variant of it, for managing the expectations for trustworthiness among cloud-enabled ubiquitous systems and web sites.

8.2 Reference Architecture
Based on the findings of our user study, even though addressing key security and privacy concerns holistically helps to minimize end-user adoption barriers, perceptions related to the trustworthiness of an IoT application hangs significantly on the implementation of security and privacy best practices in the immediately visible IoT device or application user interface. Nevertheless, with growing concerns of security and privacy in
IoT Application scenarios, we believe that the proposed reference architecture can be adopted by researchers and IoT solution architects, at large, as a yardstick for governing the implementation of security and privacy concerns at all facets of an overarching IoT solutions architecture.

9. REFERENCES


Authors

Ivor D. Addo is an IT Instructor and a PhD student at Marquette University, USA. He is also a member of ACM and IEEE. Ivor’s research interests are focused on privacy and security issues in IoT and Sociable Robotic applications. Ivor has a number of published peer-reviewed conference papers.

Sheikh Iqbal Ahamed is a professor and director of Ubicomp Lab in the department of Math., Stat., and Computer Science at Marquette University, USA. He is also a faculty member of Medical college of Wisconsin, USA. He is a senior member of the IEEE Computer Society and ACM. He completed his Ph.D. in Computer Science from Arizona State University, USA in 2003. His research interests include mHealth, security and privacy in pervasive computing and middleware for ubiquitous/pervasive computing. He has published 100+ peer reviewed journal, conference and workshop papers including nine best paper/posters. Dr. Ahamed serves regularly on international conference program committees in software engineering and pervasive computing such as COMPSAC 13, COMPSAC 12, PERCOM 08, and SAC 08. He is the Guest Editor of Computer Communications Journal, Elsevier.

Stephen S. Yau

Stephen S. Yau is Professor of Computer Science and Engineering and director of Information Assurance Center at Arizona State University (ASU). He served as the chair of Department of Computer Science and Engineering at ASU in 1994 - 2001. He was on the faculties of Northwestern University, Evanston, Illinois, and University of Florida, Gainesville. His current research is in services and cloud computing, Internet of Things, cyber security and software engineering. He served as the president of the Computer Society of the Institute of Electrical and Electronics Engineers (IEEE-CS), and was on the boards of directors of IEEE and Computing Research Association. He served as the editor-in-chief of IEEE Computer, and organized a number of major conferences, including 1989 World Computer Congress sponsored by International Federation for Information Processing, (IFIP) and IEEE Annual International Computer Software and Applications Conference. He is an IEEE Life Fellow and a Fellow of American Association for the Advancement of Science. He received the Ph.D. degree from the University of Illinois, Urbana in electrical engineering.

Arun Balaji Buduru

Arun Balaji Buduru is a Ph.D. candidate at the School of Computing, Informatics and Decision System Engineering at Arizona State University, Tempe, Arizona, USA. His research interests include cyber security, stochastic planning, and...
machine learning. He received the B.E. degree in computer science from Anna University in India.