IJBD Editorial Board

Editors-in-Chief
Ernesto Damiani, Università degli Studi di Milano, Italy
Paul Hoffman, Saffron Technology Inc., USA
Patrick Hung, University of Ontario Institute of Technology, Canada

Associate Editors
Satyendra Rana, Wayne State University, USA
Shiyong Lu, Wayne State University, USA

Editorial Board
Rafael Accorsi, University of Freiburg, Germany
Claudio Ardagna, Università degli Studi di Milano, Italy
Budak Arpınar, University of Georgia, USA
Valerio Bellandi, Università degli Studi di Milano, Italy
Rajdeep Bhowmik, Cisco Systems Inc., USA
Chiara Braghin, Università degli Studi di Milano, Italy
Suren Byna, Lawrence Berkley National Laboratory, USA
Chi-Hung Chi, CSIRO, Australia
Byron Choi, Hong Kong Baptist University, Hong Kong
Stelvio Cimato, Università degli Studi di Milano, Italy
Piero Fraternali, Politecnico di Milano, Italy
Hesham Hallal, University of Tabuk, Saudi Arabia
Peng Han, Chongqing Research Center for Information and Automation Technology (CIAT), China
Omar Hassan, Insa Lyon, France
Yongqiang He, Facebook, USA
Srividya Kona, Arizona State University, USA
Shailesh Kumar, Google India, India
Marcello Leida, EBTIC, UAE
Zhanhuai Li, Northwestern Polytechnical University, China
Du Li, Ericsson, USA
Althea Liang, HP Lab, Singapore
Carolyn McGregor, University of Ontario Institute of Technology, Canada
Hamid Motahari, University of New South Wales, Australia
Maryam Panahizar, Kno.e.sis, USA
Weining Qian, East China Normal University, China
Paul Rosen, University of Utah, USA
Gabriele Ruffatti, Engineering Ingegneria Informatica S.p.A., Italy
Ming-Chien Shan, SAP Research, USA
Zhe Shan, Manhattan College, USA
Badari Narayana Thyamagondlu Nagarajasharman, University of New South Wales, Australia
Andy Twigg, Oxford University, UK
Raymond Wong, University of New South Wales, Australia
Jian Yin, Sun Yat-Sen University, China
Gong Zhang, Oracle, USA
Jianhua Zheng, Kingdee Research, China
Zhixiong Chen, Mercy College, USA
Jia Zhang, Carnegie Mellon University - Silicon Valley, USA
International Journal of Big Data

January-March 2014, Vol. 1, No.1

Table of Contents

ASSOCIATE-EDITORS PREFACE

iv Satyendra Rana, Wayne State University, USA
Shiyong Lu, Wayne State University, USA

RESEARCH ARTICLES

1 Distributed and Scalable Graph Pattern Matching: Models and Algorithms
Arash Fard, M. Usman Nisar, John A. Miller, Lakshmish Ramaswamy,
University of Georgia, Athens, USA

15 A Throughput Driven Task Scheduler for Batch Jobs in Shared MapReduce Environments
Xite Wang, Derong Shen, Ge Yu, Tiezhang Nie, Yue Kou, Northeastern
University, P.R. China

25 Personalization Big Data vs. Privacy Big Data
Benjamin Habegger1, Omar Hasan1, Lionel Brunie1, Nadia Bennani1, Harald Kosch2,
Ernesto Damiani3
1University of Lyon, France, 2University of Passau, Germany, 3University of Milan, Italy

36 A Forecasting Approach For Data Allocation In Scalable Database Systems
Shun-Pun Li, Man-Hon Wong, The Chinese University of Hong Kong

49 Call for Articles: International Journal of Big Data (IJBD)
Welcome to the inaugural issue of International Journal of Big Data (IJBD). Big Data is fast emerging as a discipline in its own right despite its overlap with and roots in several well-established disciplines. When Big Data studies are conducted within the formulations of another field, issues germane to realizing the promise of Big Data are quite likely to be overlooked or are paid less attention. A platform dedicated to Big Data as its primary focus will go a long way in surfacing up the unique challenges and opportunities and will be instrumental in encouraging effort toward advancing state-of-the-art in Big Data.

Notwithstanding the usual scalability, performance, and modeling concerns, Big Data signifies a radical shift in the way we think about and leverage data in both science and business. IJBD is designed as a platform to provide a broader and balanced coverage of all facets of Big Data. An online journal, IJBD is published quarterly for rapid dissemination of high quality research in the Big Data field and to serve as an ongoing forum for the continuing discussion of research and ideas published in the journal.

IJBD actively seeks articles that bridge the gap between technology foundations of Big Data and the actual exploitation of Big Data. We encourage authors who have presented their papers at high quality conferences to develop an expanded version of their work for timely publication in IJBD. To ensure quality and suitability of articles to the scope of IJBD, each submitted article is reviewed by at least two IJBD Editorial Board members.

This inaugural issue is a compilation of four papers that address challenges encountered while designing a Big Data System. A Big Data system is not only influenced by the attributes of data it manages and analyzes but also by its purpose, the nature of the workload, implementation alternatives, and a slew of environmental and contextual concerns.

Extracting value from Big Data requires analysis of large inter-related data sets. Advanced analysis algorithms involve pattern matching over a distributed graph. The first article titled, “Distributed and Scalable Graph Pattern Matching: Models and Algorithms” by Arash Fard, M. Usman Nisar, John A. Miller, and Lakshmish Ramaswamy, explores distributed pattern matching problem over extremely massive graphs and propose a spectrum of efficient and scalable graph pattern matching algorithms.

Companies run Big Data jobs on a regular basis. These jobs must be completed within pre-allocated fixed time slots and with often limited resources. The second article titled, “A Throughput Driven Task Scheduler for Batch Jobs in Shared MapReduce Environments” by Xite Wang, Derong Shen, Ge Yu, Tiezhang Nie, and Yue Kou examines the effect of task scheduling on the timely execution of Big Data jobs. The jobs modeled by the study are Map/Reduce jobs in a shared environment. The authors propose a novel task scheduling algorithm.

The design challenges of a Big Data system are not just limited to scalability and performance. There are newer and conflicting demands that pertain to how information is leveraged by the next-generation of data-driven application. The third article titled, “Personalization Big Data vs. Privacy Big Data” by Benjamin Habegger, Omar Hasan, Lionel Brunie, Nadia Bennani, Harald Kosch, and Ernesto Damiani highlights such a conflicting demand by surfacing the inherent challenges of useful personalization while preserving individual privacy. Authors discuss these issues in the backdrop of EEXCESS, a concrete project aimed to both provide high level recommendations and to respect user privacy.

Associate Editor’s Preface
Design Challenges in Big Data Systems

Satyendra Rana, Wayne State University, USA
Shiyong Lu, Wayne State University, USA
Scaling is one of the key design concerns. Achieving scalability and ensuring adequate performance at the same time may require dynamic reconfiguration and reallocation of resources. The fourth article titled, “A Forecasting Approach for Data Allocation in Scalable Database Systems” by Shun-Pun Li and Man-Hon Wong, studies dynamic allocation of data fragments by predicting expecting workload. An efficient time-series based forecasting algorithm is presented and its superiority is established through simulated experimental results.

We would like to thank the authors for contributing high quality research. We would also like to thank the reviewers for their help with the review process. Finally, we are grateful for the effort Patrick Hung made in giving birth to this inaugural issue of International Journal of Big Data (IJBD).

About the Associate-Editors

**Dr. Satyendra Rana** is founding Co-Director of the Big Data & Business Analytics Group and a Visiting Professor in the College of Engineering at Wayne State University in Detroit, Michigan. Dr. Rana is an award winning entrepreneur and a veteran executive of the information technology industry with over 30 years of IT experience in various capacities. Dr. Rana brings a unique perspective and expertise that spans technology strategy development, global program implementations, customer relationships, business process innovation, enterprise architectures, IT governance, executive leadership, entrepreneurship, and academia and research. The companies founded by him have won many awards including a company of the year award by National Business Incubators Association of America. As a former educator and researcher in Computer Science, Dr. Rana supervised over 10 doctoral students and 50 master’s dissertations on various topics in Computer Science, published over 50 articles in international journals and conferences, organized conferences and sessions, and delivered many invited talks.

**Dr. Shiyong Lu** is an Associate Professor in the Department of Computer Science at Wayne State University and the Director of the Big Data Research Laboratory. Dr. Lu received his Ph.D. in Computer Science from State University of New York at Stony Brook in 2002. Before that, he received his M.E. from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing in 1996 and B.E. from the University of Science and Technology of China at Hefei in 1993. Dr. Lu's research focuses on big data and scientific workflows. He is an author of two books and over 100 articles published in various international conferences, such as IEEE International Conference on Services Computing (SCC), and international journals, such as IEEE transactions on Services Computing (TSC), and Data and Knowledge Engineering (DKE), IEEE Transactions on Knowledge and Data Engineering (TKDE), IEEE Transactions on Parallel and Distributed Systems (TPDS), IEEE Transactions on Information Technology in Biomedicine (TITB), Theoretical Computer Science (TCS), Information Systems (IS), etc. Dr. Lu's research is supported by NSF, Department of Agriculture, Michigan Tri-corridor, and Wayne State University. He is a Senior Member of the IEEE.
DISTRIBUTED AND SCALABLE GRAPH PATTERN MATCHING: MODELS AND ALGORITHMS

Arash Fard, M. Usman Nisar, John A. Miller, Lakshmish Ramaswamy

Computer Science Department
The University of Georgia
Athens, USA
{ar, nisar, jam, laks}@cs.uga.edu

Abstract

Graph pattern matching is a fundamental operation for many applications, and it is exhaustively studied in its classical forms. Nevertheless, there are newly emerging applications, like analyzing hyperlinks of the web graph and analyzing associations in a social network, that need to process massive graphs in a timely manner. Regarding the extremely large size of these graphs and knowledge they represent, not only new computing platforms are needed, but also old models and algorithms should be revised. In recent years, a few pattern matching models have been introduced that can promise a new avenue for pattern matching research on extremely massive graphs. Moreover, several graph processing frameworks like Pregel have recently sought to harness shared nothing clusters for processing massive graphs through a vertex-centric, Bulk Synchronous Parallel (BSP) programming model. However, developing scalable and efficient BSP-based algorithms for pattern matching is very challenging on these platforms because this problem does not naturally align with a vertex-centric programming paradigm. This paper introduces a new pattern matching model, called tight simulation, which outperforms the previous models in its family in terms of scalability while preserving their important properties. It also presents a novel distributed algorithm based on the vertex-centric programming paradigm for this pattern matching model and several others in the family of graph simulation as well. Our algorithms are fine-tuned to consider the challenges of pattern matching on massive data graphs. Furthermore, we present an extensive set of experiments involving massive graphs (millions of vertices and billions of edges) to study the effects of various parameters on the scalability and performance of the proposed algorithms.

Keywords: query graphs; distributed algorithms; graph simulation, pattern matching, big data

1. INTRODUCTION

Graph pattern matching, as an important class of graph queries, seeks to find subgraphs of a data graph that are similar to a given query graph. This problem has been extensively studied over the past several decades; however, the massive scales of modern application domains such as social networks and the World Wide Web have reigned interest in highly scalable graph pattern matching algorithms (Brynielsson, Hogberg, Kaati, Martenson, & Svenson, 2010) (Sun, Wang, Wang, Shao, & Li, 2012).

Subgraph isomorphism is traditionally the most popular model of graph pattern matching. While it returns the strictest matches for graph pattern matching in terms of topology (Gallagher, 2006), the problem is NP-complete (Ullmann, 1976) and thus is impractical for massive graphs. The family of graph simulation algorithms provides a practical alternative to subgraph isomorphism by relaxing its matching conditions. These new models allow matches to be found in polynomial time. Moreover, some researchers (Brynielsson, Hogberg, Kaati, Martenson, & Svenson, 2010) (Fan, Li, Ma, Tang, Wu, & Wu, 2010) (Ma, Cao, Fan, Huai, & Wo, 2011) (Fard, Abdolrashidi, Ramaswamy, & Miller, 2012) believe that graph simulation can be more appropriate than subgraph isomorphism for some modern applications such as social network analysis because it yields matches that are conceptually more intuitive.

Variants of the graph pattern matching paradigm have been proposed that form a spectrum with respect to the stringency of the matching conditions. These models include graph simulation, dual simulation, strong simulation, strict simulation, tight simulation, and subgraph isomorphism, where graph simulation is the least restrictive and strict simulation is the most restrictive (Ma, Cao, Fan, Huai, & Wo, 2011) (Fard, Nisar, Ramaswamy, Miller, & Saltz, 2013). In comparison, tight simulation is conceptually closest to subgraph isomorphism, whereas graph simulation is the farthest. It is also noteworthy that, in prior work in this area, typically the more restrictive the model, the less scalable the corresponding algorithm.

While polynomial time algorithms exist for different varieties of simulations, they still do not scale to the massive graphs that characterize modern applications such as social networks. Therefore, a natural question is whether these algorithms can be parallelized and implemented efficiently on shared nothing clusters. In recent years, several graph processing frameworks have been proposed that enable programmers to easily and efficiently implement distributed graph algorithms on shared nothing clusters. These include Pregel (Malewicz, et al., 2009), Giraph...
(Giraph website) and GPS (Salihoglu & Widom, 2012) which are characterized by the BSP programming paradigm. In these platforms, the programmer expresses algorithms in terms of the actions performed by a generic vertex of the graph. In recent years, researchers have designed vertex-centric algorithms for various graph problems. Surprisingly, to the best of our knowledge, there is no other study on vertex-centric approaches for graph simulation models. While there is some recent work on distributed basic graph simulation (Ma, Cao, Huai, & Wo, 2012), the proposed algorithm is not based on the vertex-centric programming paradigm.

Considering the unique benefits offered by vertex-centric programming platforms, this paper is dedicated to exploring the following questions. (1) How well do graph simulation in general, and various simulation models in particular fit into the vertex-centric distributed processing paradigm? (2) If certain simulation models are not suitable for the vertex-centric programming paradigm, what are the major bottlenecks? (3) Can we develop new simulation models that are conceptually similar to the existing ones, but are better suited to the vertex-centric paradigm? (4) How do various graph properties such as the number of vertices and the density of a graph impact the performance and scalability of the distributed graph simulation algorithms?

Towards answering these questions, we have designed vertex-centric distributed algorithms for various graph simulation models. We have comprehensively studied their behaviors through an extensive set of experiments. The main technical contributions of this paper are summarized below.

- We identify the bottlenecks that severely limit the scalability and performance of strong and strict simulation on distributed graph processing platforms. Towards ameliorating these limitations, we propose a new graph simulation model called *tight simulation*. Tight simulation is vastly more amenable to vertex-centric distribution, and surprisingly, the quality of its result is the same or in some cases better than both strong and strict simulation.

- We propose novel vertex-centric distributed algorithms for all five simulation models. Our algorithms are based upon detailed analytical studies of the models, and they seek to mitigate various performance bottlenecks.

- We also present a detailed experimental study on the benefits and costs of the proposed distributed algorithms for various graph simulation models. The experiments validate that our tight graph simulation model is highly scalable and very well suited for vertex-centric distribution.

The rest of the paper is organized as follows. In section two, we briefly review concepts of BSP-based vertex-centric graph processing and graph pattern matching models.

2. BACKGROUND

In this section, we briefly review concepts of BSP-based vertex-centric graph processing and graph pattern matching models.

2.1. BSP vertex-centric Graph Processing

Several distributed computing platforms have been recently designed and implemented with the goal of effectively harnessing shared nothing clusters for processing massive graphs. The most popular platforms among them are based upon BSP-based vertex-centric programming model, like Pregel (Malewicz, et al., 2009), GPS (Salihoglu & Widom, 2012) and Giraph (Giraph website).

Bulk Synchronous Parallel (BSP) was first proposed by Valiant (Valiant, 1990) as a model of computation for parallel processing. In this model, computation is a series of *supersteps*. Each superstep consists of three ordered stages: (1) concurrent computation, (2) communication, and (3) barrier synchronization. In the vertex-centric programming model, each vertex of the data graph is a computing unit which can be conceptually mapped to a process in the BSP model. At loading time, the data graph is partitioned among several *workers*. Each worker is a real process which handles the computation for a partition of vertices. Each vertex initially knows only about its own label and its outgoing edges. Vertices can then exchange messages through successive supersteps to learn about their neighborhood or to accomplish a computing task. Within each superstep, vertices are executing the same predefined function. A vertex votes to halt and goes to inactive mode when it believes that it has accomplished its tasks. It remains inactive until it is triggered externally by a message from another vertex. The algorithm terminates when all vertices become inactive.

The vertex-centric approach is a pure message passing model in which programmers focus on a local action for each vertex of the data graph. The usage of the BSP model makes it inherently free of deadlocks. Moreover, it is well-suited for distributed implementation by design and can provide very high scalability.

2.2. Pattern Matching Models

A pattern matching algorithm tries to find all the matches of a given graph, called a query graph, in an existing larger graph, called a data graph. To define it more formally, assume that there is a data graph \( G(V, E, l) \), where \( V \) is the set of vertices, \( E \) is the set of edges, and \( l \) is a function that maps the vertices to their labels. Given a query graph \( Q(V_q, E_q, l_q) \) the task is to find all subgraphs of \( G \) that
match the query $Q$. Here, we assume all vertices are labeled, all edges are directed, and there are no multiple edges. Without loss of generality, we also assume a query graph is a connected graph because the result of pattern matching for a disconnected query graph is equal to the union of the results for its connected components. In this paper, we use pattern and query graph interchangeably.

**Subgraph Isomorphism.** Subgraph isomorphism is the most famous model for pattern matching. It preserves all topological features of the query graph in the result subgraph. However, finding all subgraphs that are isomorphic to a query graph is an NP-hard problem in the general case (Ullmann, 1976). By definition, subgraph isomorphism describes a bijective mapping between a query graph $Q(V_q, E_q, I_q)$ and a subgraph of a data graph $G(V, E, I)$, denoted by $Q \simeqiso G$.

**Graph Simulation.** Graph simulation, provides a faster algorithm by relaxing some restrictions on matches.

**Definition:** Pattern $Q(V_q, E_q, I_q)$ matches data graph $G(V, E, I)$ via graph simulation, denoted by $Q \simeqsim G$, if there is a binary relation $R \subseteq V_q \times V$ such that (1) if $(u, u') \in R$, then $l_q(u) = 1(u')$; (2) $\forall u \in V_q, \exists u' \in V: (u, u') \in R$; (3) $\forall (u, v) \in E_q, \exists (u', v') \in E: (u, u') \in R \land (v, v') \in R$.

Intuitively, graph simulation only preserves the child relationships of each vertex. The result of pattern matching is a maximum match set of vertices. The maximum match set, $R_m \subseteq V_q \times V$, is the biggest relation set between $Q$ and $G$ with respect to $Q \simeqsim G$. The result match graph, $G_r(V_r, E_r, I_r)$, is a subgraph of $G$ that can represent $R_m$ (Ma, Cao, Fan, Huai, & Wo, 2011). By definition, $G_r$ is a subgraph of $G$ which satisfies these conditions: (1) $(u, u') \in R_m \iff u' \in V_r$; (2) $\forall (u, u'), (v, v') \in R_m \iff (u, v) \in E_q \iff (u', v') \in E_r$.

A quadratic time algorithm for graph simulation was first proposed in (Henzinger, Henzinger, & Kopke, 1995) with applications to the refinement and verification of reactive systems. This model and its extensions have been studied especially in recent years (Fan, Li, Ma, Tang, Wu, & Wu, 2010), (Ma, Cao, Huai, & Wo, 2012) because of their new applications in analysis of social networks (Brynielsson, Hogberg, Kaati, Martenson, & Svenson, 2010).

**Dual Simulation.** Dual simulation improves the result of graph simulation by taking into account not only the children of a query node, but also its parents.

**Definition:** Pattern $Q(V_q, E_q, I_q)$ matches data graph $G(V, E, I)$ via dual simulation, denoted by $Q \simeqsim G$, if (1) $Q \simeqsim G$ with a binary match relation $R_D \subseteq V_q \times V$, and (2) $\forall (u, u') \in R_D \Rightarrow \exists w' \in V: (w, w') \in E_q \land (w', u') \in E$.

In dual simulation similar to graph simulation, we maintain the concept of a maximum match set and a result match graph. Here, the result match graph is also a single graph which might be connected or disconnected. In (Ma, Cao, Fan, Huai, & Wo, 2011) a cubic algorithm for dual simulation is proposed.

**Strong Simulation.** Strong simulation adds a locality property to dual simulation. Shuai Ma et al. (Ma, Cao, Fan, Huai, & Wo, 2011) introduce the concept of a ball to define locality. A ball $b$ in $G(V, E, I)$, denoted by $G[v, r]$, is a subgraph of $G$ that contains all vertices not further than a specified radius $r$ from a center $v \in V$; moreover, the ball contains all edges in $G$ that connect these vertices (i.e., it is an induced connected subgraph). Given two vertices $u$ and $v$ in a connected graph, the distance between them is defined as the minimum number of edges in an undirected path which connects them. The diameter of a connected graph is then defined as the longest distance between any pair of vertices in the graph.

**Definition:** Pattern $Q(V_q, E_q, I_q)$ matches data graph $G(V, E, I)$ via strong simulation, denoted by $Q \simeqsim G$, if there exists a vertex $v \in V$ such that (1) $Q \simeqsim G[v, d_q]$ with maximum dual match set $R_D^b$ in ball $b$ where $d_q$ is the diameter of $Q$, and (2) $c$ is member of at least one of the pairs in $R_D^b$. The connected part of the result match graph of each ball with respect to its $R_D^b$ which contains $c$ is called a maximum perfect subgraph of $G$ with respect to $Q$.

In contrast to the previous types of simulation, strong simulation may have multiple maximum perfect subgraphs (Max-PGs) as its result. As indicated by (Ma, Cao, Fan, Huai, & Wo, 2011), although strong simulation preserves many of the topological characteristics of a pattern graph, its result can be computed in cubic time. Moreover, the number of MaxPGs is bounded by the number of vertices in a data graph, while subgraph isomorphism may have an exponential number of match subgraphs.

**Comparing the result of the models.** Figure 1 provides an example to show the difference in the results of the mentioned pattern matching models. This example is about finding a team of specialists on a social network like LinkedIn, inspired by an example in (Ma, Cao, Fan, Huai, & Wo, 2012). A vertex in this example represents a member, and a label represents a member's profession. A directed edge from a member $a$ to a member $b$ indicates that member $a$ has endorsed member $b$. Because the vertices in the query graph for this example have distinct labels, we only specify data graph vertex ID's for a match rather than specifying the full match relation. Given the pattern and the data graph, the maximum match set for graph simulation consists of all vertices except $\{2,13\}$. It is clear that vertices 1, 3, and 14 are not appropriate matches either. Dual simulation removes these inappropriate vertices; its maximum match set contains all vertices except $\{1,2,3,13,14\}$. However, there are still some vertices that do not provide very meaningful
matches. For example, the cycle \( \{15,16,17,18,19,20\} \) creates a very big subgraph match, which is not desirable for such a small pattern. Applying strong simulation then shrinks the result match graph to a more appropriate set of vertices; the result here is the set of vertices \( \{4,5,6,7,8,9,10,11,12\} \). In contrast, there are two isomorphic subgraphs corresponding to these two set of vertices: \( \{4,6,7,8\} \) and \( \{5,6,7,8\} \).

**Eccentricity, center, and radius.** The eccentricity of a vertex in a graph is its maximum distance from any other vertex in the graph. The vertices of the graph with the minimum eccentricity are the centers of the graph, and the value of their eccentricity is the radius, \( r_Q \), of the graph. The maximum value of eccentricity equals to the diameter of the graph \( d_Q \). It is proved that \( r_Q \leq d_Q \leq 2r_Q \) (Farber, 1989).

3. **Strict simulation**

Strict simulation is a novel modification of Strong simulation that not only substantially improves its performance, but also maintains a better quality of result because of its revised definition of locality. The locality restriction defined in strong simulation is the main reason for its long computation time because the size of balls can be potentially very big. Here, the size of a ball means the number of vertices \( |Q| \) that it contains. Proportional to the diameter of \( Q \) and average degree of vertices in \( G \), each ball in strong simulation can be fairly bulky. Furthermore, due to communication overhead during ball creation, the problem is exacerbated in distributed systems where a data graph is partitioned among different nodes. In order to mitigate this overhead, we introduced a pattern matching model, named **strict simulation** in (Fard, Nisar, Ramaswamy, Miller, & Saltz, 2013). Strict simulation is more scalable and preserves the important properties of strong simulation. It is shown in (Ma, Cao, Fan, Huai, & Wo, 2011), (Fan W., 2012) that: (1) if \( Q \subset_{iso} G \), then \( Q \subset_{sim}^2 G \); (2) if \( Q \subset_{sim}^0 G \), then \( Q \subset_{sim}^1 G \); and (3) if \( Q \subset_{sim}^0 G \), then \( Q \subset_{sim}^1 G \). We show that strict simulation and another new model we introduce in the next section are not only more efficient than strong simulation, but also more stringent; i.e., their results are getting closer to subgraph isomorphism while they become computationally more efficient.

Figure 3.a shows the flowchart of the centralized algorithm for strong simulation. In this algorithm, the match relation of dual simulation, \( R_D \), is computed first. Then, a ball \( G_D[v,d_Q] \) is created for each vertex \( v \) of the data graph contained in \( R_D \). Members of the ball are selected regardless of their membership in pairs of \( R_D \). At the next step, the dual match relation is projected on each ball to compute the result of strong simulation. Finally, the maximum perfect subgraph (MaxPG) is extracted by constructing the match graph on each ball and finding the connected component containing the center.

In comparison, Figure 3.b illustrates the centralized algorithm for strict simulation. The key difference between them is that in strict simulation the duality condition is enforced before the locality condition; i.e., balls are created from the dual result match graph, \( G_D \), rather than from the original graph. As the balls in strict simulation are often significantly smaller, this seemingly minor difference between the two algorithms has a profoundly positive impact on running time.

There are the same numbers of balls in strict and strong simulation; however, the extracted MaxPG from each ball in strict simulation is always a subgraph of the result in strong simulation on a ball with the same center. Duplicate MaxPGs may be produced from different balls, and in the case of strict simulation a result MaxPG might be subgraph of another one. In a post-processing phase, the duplicate results are filtered, and only the smaller result is kept when it is a subgraph of another result. Any possible isomorphic subgraph match of the query will be preserved in the result of strict simulation.

Figure 2 shows an example that highlights the difference between the results of these two types of simulation. In this example, all vertices except 9 will appear in the dual result match graph. For strong simulation, a ball centered at vertex 2 would contain all the vertices in the data graph. In contrast, because the dual match graph would not contain vertex 9 and some of the edges, the corresponding ball for strict simulation would only contain the set of vertices \( \{1,2,3,4\} \). Therefore, the MaxPG resulting from strict simulation will contain all vertices except 9, while the one resulting from strict simulation will contain only vertices 1, 2, and 3. One can verify that this is the only MaxPG which will result from any ball of strict simulation.
The formal definition of strict simulation follows.

**Definition:** Pattern \( Q(V_q, E_q, l_q) \) matches data graph \( G(V, E, l) \) via strict simulation, denoted by \( Q \preceq_{\text{str}} G \), if there exists a vertex \( v \in V \) such that (1) \( v \in V_D \) where \( G_D(V_D, E_D, I_D) \) is the result match graph with respect to \( Q \preceq_{\text{str}} G \); (2) \( Q \preceq_{\text{str}} G_D[v,d_q] \) where \( G_D[v,d_q] \) is a ball extracted from \( G_D \), and \( d_q \) is the diameter of \( Q \); (3) \( v \) is a member of the result MaxPG.

A MaxPG for strict simulation is defined the same as a MaxPG for strong simulation. Using the definition and the properties of dual simulation presented in (Ma, Cao, Fan, Huai, & Wo, 2011), (Fan W., 2012), properties of strict simulation can be proved as follows.

**Theorem 1:** For any query graph \( Q \) and data graph \( G \) such that \( Q \preceq_{\text{str}} G \), there exists a unique set of maximum perfect subgraphs for \( Q \) and \( G \).

Proof: It is proved that the result of dual simulation is unique. Therefore, the balls extracted from its result match graph and consequently their result after applying dual filter would be unique.

**Proposition 1:** (1) If \( Q \preceq_{\text{iso}} G \), then \( Q \preceq_{\text{str}} G \). (2) if \( Q \preceq_{\text{str}} G \), then \( Q \preceq_{\text{iso}} G \).

Proof: (1) Any subgraph isomorphism match in \( G \) is also dual match to \( Q \); therefore, it will appear in the result of dual match graph. Clearly, a ball with radius \( d_q \) (diameter of \( Q \)), and centered on a vertex of this subgraph will contain its whole vertices. The subgraph later will also be in the result of dual filter on the ball. (2) \( G_D \) is a subgraph of \( G \), and the distance between any pair of vertices in \( G_D \) is smaller than their distance in \( G \). Therefore, any ball in strict simulation will be a subgraph of the corresponding ball in strong simulation with the same center. Consequently, when there is a subgraph result in the ball of strict simulation after applying dual filter, it will be also preserved in the result of strong simulation.

**Proposition 2:** The number of maximum perfect subgraphs produced by \( Q \preceq_{\text{str}} G \) is bounded by the number of vertices in \( G \).

Proof: \( G_D \) is a subgraph of \( G \), and the number of balls in strict simulation equals to the number of vertices in \( G_D \). Moreover, not more than one MaxPG can be produced from each ball. Therefore, their total number is bounded by the number of vertices in \( G \).

Theorem 1 ensures that strict simulation yields a unique solution for any query/data graph pair. Proposition 1 indicates that strict simulation is a more stringent notion than strong simulation, though still looser than subgraph isomorphism. Proposition 2 gives an upper bound on the number of possible matches for a query graph in a data graph. It should be noted that the number of matches in subgraph isomorphism can be exponential to the number of vertices in \( G \). The following theorem also shows that the asymptotic time complexity of strict simulation is the same as that of strong and dual simulation.

**Theorem 2:** For any query graph \( Q \) and data graph \( G \), the time complexity for finding all maximum perfect subgraphs with respect to strict simulation is cubic.

Proof: It is proved that time complexity for finding \( G_D \) is cubic. Moreover, it is proved that time complexity of strong simulation is cubic. Regarding the fact that each ball in strict simulation is a subgraph of its corresponding ball in strong simulation, time complexity of strict simulation is also cubic.

### 4. Tight Simulation

Strict simulation reduces the computation time of Strong simulation by decreasing the size of the balls, but the number of balls remains the same. The number of balls can be big when the number of vertices left after of dual simulation is big which exacerbates the performance. Moreover, it is still desirable, both in terms of computation time and the quality of the result, to shrink the size of the balls. In this section, we introduce a new novel pattern matching model, named tight simulation, which not only decreases the size of the balls further in comparison to strict simulation, but also reduces the number of balls. This will make the search for the pattern faster and the resulting subgraphs more stringent.

A summary of the centralized algorithm for tight simulation is compared to strict simulation in Figure 3.c. The main difference between these two is on the phase of preprocessing the query graph, and ball creation. In the phase of preprocessing, a single vertex, \( u \in Q \), is chosen as a candidate match to the center of a potential ball on the data graph. The appropriate radius of the ball is also calculated in this phase. Then, in the phase of ball creation, only those vertices of the data graph which are in the dual match set of \( u \) will be picked as the center of balls. The pre-calculated radius of such a ball is always between the radius and the diameter of the query graph. It should be noticed...
that only the vertices in the result of dual match graph will be used for ball creation, similar to strict simulation.

We introduce multiple selectivity criteria for finding the candidate vertex and radius out of the query graph. A vertex $u \in Q$ with the minimum eccentricity (a center of $Q$) which has the highest ratio of degree to label frequency (in $Q$) will be picked as the candidate vertex. Selecting one of the centers of $Q$ as the candidate vertex makes it also possible to select the candidate radius of balls equal to the radius of $Q$. It is the tightest ball which also preserves all the subgraph isomorphic matches of the query graph in the result. Among the potential vertex candidates those with the highest degree and lowest label frequency present higher selectivity condition. In the case that there are several vertices with same selectivity score, one of them will be selected randomly.

In a centralized algorithm, it is possible to postpone selecting the candidate vertex and radius until the end of dual simulation phase and right before ball creation. One may consider that a vertex $u \in Q$ will be picked as the candidate vertex. Selecting one of the vertices $u \in Q$ as the center of balls with radius $r_Q$ will be the best vertex candidate. In this case, the candidate radius would be equal to the eccentricity of $u$ in $Q$. Although this choice might be a good option in a centralized algorithm, it is not viable for a distributed algorithms based on vertex-centric framework. In such a distributed environment every vertex of $G$ will learn if it is a member of a dual match set to a few vertices in $Q$, but no global view of this set would be available. Proposed vertex-centric distributed algorithms will be explained in the next section.

The results of tight simulation are subgraphs of the corresponding results of strict simulation while they always contain all the subgraph isomorphic matches. Therefore, the results of tight simulation are closer to subgraph isomorphism in comparison to strict simulation. It should be noted that the post-processing phase explained in the last section will be applied to the results of strong, strict, and tight simulation in the same way. Figure 4 shows an example which displays the difference between the results of tight versus strict and strong simulation. In this example, all the vertices shown in the data graph will remain in the dual match graph. Clearly, the vertex labeled $B$ in the pattern is its center; hence, will be picked as the candidate vertex. Therefore, vertices $\{2,4,6,10,12,14\}$ will be picked as the center of balls with radius $r_Q = 1$ in tight simulation. Only the ball centered at 2 can result a MaxPG which contains these vertices $\{1,2,3\}$. In contrast in strict and strong simulation, a ball with radius $d_Q = 2$ will be created for any vertex of the data graph. One can verify that for the ball created on vertex 1, strict simulation results a MaxPG containing $\{1,2,3,4,5,6,7,8\}$, and strong simulation results a MaxPG containing all the vertices.

We formally define tight simulation as follows.

**Definition:** Pattern $Q(V_q, E_q, l_q)$ matches data graph $G(V, E, l)$ via tight simulation, denoted by $Q \preceq T_{sim} G$, if there are vertices $u \in Q$ and $u' \in G$ such that (1) $u$ is a center of $Q$ with highest defined selectivity; (2) $(u, u') \in R_D$ where $R_D$ is dual relation set between $Q$ and $G$; (3) $Q \preceq T_{sim} G[D[u', r_Q]]$ where $D[u', r_Q]$ is a ball extracted from $G_D(V_D, E_D, l_D)$ which is the result match graph with respect to $Q \preceq T_{sim} G$, and $r_Q$ is the radius of $Q$; (4) $u'$ is a member of the result MaxPG.

![Figure 4. Comparing tight vs. strict and strong simulation](image)

The criterion for selectivity of $u$ in $Q$ is the ratio of its degree to its label frequency. The definition of MaxPG is also similar to its definition for strong and strict simulation. Similar to strict simulation, we can assert and prove the properties of tight simulation as follows.

**Theorem 3:** For any query graph $Q$ and data graph $G$ such that $Q \preceq T_{sim} G$, there exists a unique set of maximum perfect subgraphs for $Q$ and $G$.

**Proof:** It is proved that the result of dual simulation is unique. The candidate vertex selected from the query is also unique when the query and the selectivity criteria are fixed. Therefore, the balls created in tight simulation and their result after dual filter will be also unique.

**Proposition 3:** (1) If $Q \preceq_{iso} G$, then $Q \preceq T_{sim} G$. (2) if $Q \preceq T_{sim} G$, then $Q \preceq_{iso} G$.

**Proof:** (1) Any subgraph isomorphic match in $G$ is also dual match to $Q$; therefore, it will appear in the result of dual match graph. The dual match to candidate vertex of $Q$ is one of the vertices of the isomorphic match and will be selected as the center of a ball with radius $r_Q$. As the candidate vertex was the center of $Q$, the isomorphic match will be entirely enclosed in the ball and therefore will appear in the result of tight simulation. (2) When there is a subgraph result in the ball of strict simulation, it will also appear as a part of the result of strict simulation because there is a corresponding ball in strict simulation created on $G_D$ with a bigger radius. In other words, a ball in tight simulation is always a subgraph of its corresponding ball in strict simulation.

**Proposition 4:** The number of maximum perfect subgraphs produced by $Q \preceq T_{sim} G$ is bounded by the number of vertices in $G$. 
Proof: The results of tight simulation are subset of the results of strict simulation. The number of maximum perfect subgraphs produced by \( Q \preceq_{sims} G \) is bounded by the number of vertices in \( G \); hence, it is the same for \( Q \preceq_{sims} G \).

**Theorem 4:** For any query graph \( Q \) and data graph \( G \), the time complexity for finding all maximum perfect subgraphs with respect to tight simulation is cubic.

Proof: The time complexity for finding the center and the radius of \( (V_q, E_q, l_q) \) is \( \theta(|V_q|^3) \). The procedure of tight simulation is similar to strict simulation, but it deals with a smaller number of balls which are most likely smaller than the corresponding balls in strict simulation; therefore, its time complexity must be smaller as well. Taking into account that the time complexity of dual simulation is cubic, we can conclude that it is the same for tight simulation.

## 5. DISTRIBUTED PATTERN MATCHING

In contrast to the usual graph programming models, an algorithm in the vertex-centric programming model should be designed from the perspective of each vertex of a graph.

In this section, we present distributed algorithms for different types of graph simulation based vertex-centric programming model.

### 5.1. Distributed Graph Simulation

**Superstep 1:**

- Set match flag true when a vertex in the query has the same label
- Make a local matchSet of potential match vertices
- Ask children about their status
- Otherwise vote to halt

**Superstep 2:**

- If received a message while the flag is true reply back with matchSet
- Otherwise vote to halt

**Superstep 3:**

- If match flag is true evaluate the members of matchSet
  - In the case of any removal from matchSet, inform parents and set match flag accordingly
  - Otherwise vote to halt
- Otherwise vote to halt

**Superstep 4 and beyond:**

- If there is any incoming removal message reevaluate matchSet
  - In the case of any removal from matchSet, inform parents and set match flag accordingly
  - Otherwise vote to halt
- Otherwise vote to halt

Figure 5 shows a summary of our algorithm for a vertex in distributed graph simulation. Initially, we distribute the query graph among all workers. The cost of this distribution is negligible because the size of query is small and the total number of workers is limited to the number of processing elements in the system. Different tasks in different supersteps are distinguished using an if-else ladder. Moreover, the BSP framework ensures that all vertices are always at the same superstep.

A boolean flag, named match, is defined for each vertex in \( G \) in order to track if it matches a vertex in \( Q \). It is initially assumed that the vertex is not a match. Then at the first superstep, the match flag becomes true if its label matches the label of a vertex in \( Q \). In this case, a local match set, named matchSet, is created to keep track of its potential matches in \( Q \). Each vertex, then, learns about the matchSets of its children during the first three supersteps and keeps them in a local list for later evaluation of graph simulation conditions. Any match is removed from the local matchSet if it does not satisfy the simulation conditions. The vertex should also inform its parents about any changes in its matchSet. Consequently, any vertex that receives changes in its children’s matchSets reflects those changes in its list of match children and reevaluates its own matchSet. The algorithm can terminate after the third superstep if no vertex removes any match from its matchSet. This procedure will continue in superstep four and beyond until there is no change. To guarantee the termination of the algorithm, any active vertex with no incoming message votes to halt after the third superstep. At the end, the local matchSet of each vertex contains the correct and complete set of matches between that vertex and the vertices of the query graph.

Figure 6 displays an example for distributed graph simulation. Here, all the vertices of the data graph labeled a, b, and c make their match flag true at the first superstep, and then vertices 1, 2, and 5 send messages to their children. At the second superstep only vertices 5, 6, and 7 will reply back to their parents. At the third superstep, vertices 1, 5, 6, 7, and 8 can successfully validate their matchSet, but vertex 2 makes its flag false, because it receives no message from any child. Therefore, vertex 2 sends a removal message to vertex 1. This message will be received by vertex 1 at superstep four. It will successfully reevaluate its match set, and the algorithm will finish at superstep five when every vertex has voted to halt (there is no further communication).

**Figure 5. Distributed graph simulation algorithm**

**Figure 6. An example for distributed graph simulation**

The correctness of the proposed distributed algorithm can be derived from the following lemmas.

**Lemma 1:** The proposed distributed algorithm for graph simulation will eventually terminate.

Proof: The algorithm will terminate when all the vertices vote to halt and become inactive. After the third superstep, only the vertices are active which have received removal messages. A removal messages is sent from a vertex to its parents when it removes a member of its matchSet. The total
number of members of all matchSets is finite; therefore, the algorithm will terminate eventually in a finite time when all the matchSets become empty in the worst case.

**Lemma 2:** At the end of the proposed algorithm, the matchSet of each vertex contains the correct and complete set of matches for that vertex.

Proof: At the first superstep, each vertex creates its matchSet from any vertex in Q with the same label. Hence, any potential match initially becomes a member of this set. The set is filtered during the next supersteps, and it is expected that it will contain only correct matches at the end. In other words, the completeness condition of the set is satisfied at the first superstep, and we should only prove the correctness of its members when the algorithm terminates.

BSP computational model ensures that all vertices are synchronized at the beginning of each superstep. Having this property in mind, the set of supersteps 4 and beyond in the proposed algorithm (Figure 5) is very similar to a while loop. Therefore, this lemma can be proved using loop invariant theorem (Hoare, 1969). Here, the invariant is the validation of each matchSet with respect to the local list of match children. At the end of the third superstep, each vertex has a matchSet which its members are validated based on the information gathered from the matchSet of its children. The guard condition for iterating through supersteps 4 and beyond is receiving at least one removal message. The invariant condition is true at the beginning of each superstep, and will be also true at the end of the superstep because the vertices that have received any removal message will update their list of match children accordingly and reevaluate their matchSets. According to lemma 1, the guard condition will become false after a finite number of iterations. Existence of no removal message means that all vertices have satisfied the children condition. Therefore, the invariant is true after termination; i.e., each member of a matchSet is a correct match.

Figure 7 demonstrates the number of supersteps in the worst case. One can verify that if the length of path 1,2,3,4 is increased to \( L_p \) in such a way that the label pattern of vertices 3 and 4 are repeated, the algorithm will need \( L_p + 1 \) supersteps to terminate. In general, the minimum number of supersteps is 3, and its upper bound is \( O(|E|) \).

![Figure 7](image)

*Figure 7. An example for the number of supersteps*

5.2. Distributed Dual Simulation

The distributed algorithm for dual simulation is a smart modification of the distributed algorithm proposed for graph simulation in the previous subsection. Indeed, we extend the algorithm to check parent relationship as well. Therefore, each vertex also needs to keep track of the matchSets of its parents.

At the first superstep each vertex sends not only its ID, but also its label to its children. At the second superstep a vertex can infer the matchSets of its parents from the received labels and store them. Having this initial list at the second superstep allows each vertex to verify the parent relationships for each of the candidate matches in its matchSet. Very similar to the idea explained in the previous subsection for child relationships, removals from matchSet caused by evaluating the parent relationship must be reported to the children. The rest of the algorithm remains similar to the algorithm for graph simulation, with a few small modifications to consider the evaluation of a vertex with respect to its parent relationships.

The proof of correctness for this algorithm is very similar to the proof of correctness for the graph simulation algorithm. Similarly, the upper bound on the number of required supersteps for the distributed algorithm of dual simulation is \( O(|E|) \).

5.3. Distributed strong, strict, tight simulation

The algorithms for distributed strong, strict, and tight simulation are built on top of the algorithm for distributed dual simulation. In the case of strong and strict simulation, each vertex that has successfully passed the filter of dual simulation will create a ball around itself. Recalling their definitions, each ball in strong simulation is an induced connected subgraph of the data graph; whereas, each ball in strict simulation is an induced connected subgraph of the dual match graph. In the case of tight simulation, only those vertices that find themselves a dual match of the candidate vertex of \( Q \) will create a ball around itself.

The distributed algorithms that we have designed and implemented for strong and strict simulation follow the flowcharts of Figure 3, but in a distributed fashion. The first step for applying dual simulation is the same as the distributed algorithm for dual simulation. At the end of the dual simulation phase, each vertex of the data graph has a matchSet that contains the IDs of vertices of \( Q \) that match to that vertex. Any vertex with a non-empty matchSet which is considered as a member of a ball in a breadth-first search (BFS) fashion. Ball creation phase takes \( 2(R − 1) \) supersteps where \( R \) is the selected radius for the ball. For strong and strict simulation \( R = d_Q \), while it is \( R = r_Q \) for tight simulation. Moreover, all the vertices in the neighborhood are considered as members of a ball in strong simulation. In contrast for strict and tight simulation, only vertices with the match flag set to true will answer the request for neighborhood information. In the latter case, while the center vertex of the ball receives neighborhood information, it adds a vertex to the ball only if there is a corresponding edge in the pattern graph. Eventually, the center vertex of each ball will perform the rest of the computation on that ball in a sequential fashion.
Because vertices are distributed among workers, this phase can be considered an embarrassingly parallel workload.

6. Experimental Study

This section is dedicated to experimental study which aims to evaluate the new pattern matching models and the proposed distributed algorithms. Regarding the distributed algorithms, the study attempts to learn about their bottlenecks and design trade-offs with respect to the properties of their inputs. We implemented our distributed algorithms on the GPS platform (Salihoglu & Widom, 2012), which is similar to Google’s proprietary Pregel system.

The parameters for data graphs are the number of vertices, denoted by $|V|$, the density of the graph, denoted by parameter $\alpha$, where $|E| = |V|^\alpha$, and the number of distinct labels, denoted by $l$. In all experiments, $l = 200$, unless it is mentioned explicitly. The parameters for queries are also the number of vertices, denoted by $|V_q|$. Another parameter in the experiments is the number of workers denoted by $k$.

6.1. Experimental setting

We used both real world and synthesized datasets in our experiments. In terms of real world datasets, we used uk-2002 with 18,520,486 vertices and 298,113,762 edges; ljournal-2008 with 5,363,260 vertices and 79,023,142 edges; and amazon-2008 which has 735,323 vertices and 5,158,388 edges (Datasets) (Boldi & Vigna, 2004).

We used graph-tool (graph-tool website) to synthesize small and medium size randomly generated data graphs, but because of its memory limits we also implemented our own graph generator to synthesize large semi-randomly generated graphs. The input parameters of our graph generator are the number of vertices, the average number of outgoing edges $d_{ego}$, and the number of distinct labels. It picks a random integer between 0 and $2d_{ego}$ as the number of outgoing edges for every vertex. Then, for each outgoing edge, the endpoint of the edge is randomly selected. The label of each vertex is also a randomly picked integer number between 1 and $l$.

To generate a pattern graph, we randomly extract a connected subgraph from a given dataset. Our query generator has two input parameters: the number of vertices and the desired average number of outgoing edges. Unless mentioned otherwise, the average number of outgoing edges is set to such a value that $\alpha = 1.2$.

The experiments were conducted using GPS on a cluster of 8 machines. Each one has 128GB DDR3 RAM, two 2GHz Intel Xeon E5-2620 CPUs, each with 6 cores. The intra-connection network is 1Gb Ethernet. One of the machines plays the role of master.
6.2. Experimental results

The results of the experiments are categorized in four groups. It should be mentioned that distributed strong simulation is so slow on large datasets that we could run it only on fairly small datasets to be compared with strict and tight simulation. It is also noteworthy that presenting running time or speedup of different models in the same chart is only for studying their scalability. In other words, when the quality of pattern matching increases from graph simulation to dual and then strong simulation, the running time also increases. Strict and tight simulation are exceptions; i.e., we observe decrease in running time when the quality increases from strong to strict and then tight simulation.

We also performed a set of experiments to compare the quality of the results of strong, strict, and tight simulation. For comparison, we measured a few parameters in their set of subgraph results including: the number of subgraph results, their total number of distinct vertices, their total number of distinct edges, and the average and standard deviation of their diameters. We found that the number of subgraph results is increasing and their diameter is decreasing while we change the model from strong to strict and from strict to tight simulation. However, our experiments have yet shown no significant difference in either the total number of distinct vertices or the total number of distinct edges.

**Experiment 1 - Running time and Speedup.** We examine the running time and speedup of the proposed algorithms to study their performance and scalability (Figure 8). It can be observed that the running time of tight simulation is always less than the running time of strict simulation.

In the experiment displayed in Figure 8.a, we could not run the test on a single machine because of memory limits. Therefore, we extrapolate its running time.

As expected, the BSP model scales very well on bigger datasets. Moreover, all types of simulation exhibit a filtering behavior, meaning that they start by processing a large set and then refine it; this causes light workload at the final supersteps.

**Experiment 2 - Impact of pattern.** Figure 9 shows that running time increases as the size of the query becomes bigger, which is not surprising. The behavior remains similar across datasets with different numbers of vertices. The running times of strict and tight simulations are similar for small patterns because the difference in the overhead of ball-creation phase is negligible. However, it can be observed that the difference between their running-time increases with increase in the size of pattern.

**Experiment 3 - Impact of dataset.** In the first experiment of this group (Figure 10.a), we compare the running time of different pattern matching models with...
respect to the number of vertices in the data graph. As expected, the running time increases with growth in the size of data graph. The ratio between the running times of graph simulation and dual simulation shows the difference between their computational complexities. The increasing difference between the running times of strict and dual simulation can be explained by the fact that the number of balls increases proportionally to the number of vertices in the result of dual simulation. However, the rate of increase in difference of tight and strict simulation is very smaller because even in the case of tight simulation all the active vertices after dual-filtering phase will contribute to the ball creation although the number of balls is smaller as well.

Figure 10.c shows the total number of supersteps for the same set of experiments. It is not surprising that tight simulation needs less number of supersteps to terminate because the radius of its ball is smaller. The stable number of supersteps indicates the scalability of the algorithms with respect to the number of vertices in data graph. That is, increase in the size of the data graph mostly increases the local computation of the workers not their communication.

Figure 10.b shows the impact of the density of a data graph on running time. An increase in running time is to be expected. The unchanged cost of ball creation (difference between dual, strict and tight simulation) reveals a good feature of strict and tight simulation; because the density of the data graph does not have a big impact on the density of the dual match graph, the balls do not necessarily increase in size as the density of the data graph increases. The total number of supersteps for these experiments is reported in Figure 10.d. The small changes in the number of supersteps indicate the scalability of the algorithms with respect to the density of data graph.

**Experiment 4- Comparison of strong, strict, and tight simulation.** The difference in behavior between the algorithms for strong, strict, and tight simulation can be seen in Figure 11. Because of the high cost of ball creation in distributed strong simulation, it was only possible to test it on fairly small datasets.

Figure 11.a and Figure 11.b compare the running time of the three algorithms on two different datasets and a range of query sizes. The differences between strong simulation and the other two are huge on both datasets, though they are small for \(|V_q| \leq 10\). The running time of tight simulation is always slightly better than strict simulation.

Figure 11.c shows the total size of the communication in the system per superstep for strong, strict, and tight simulation. The chart shows three phases for the procedure. The first phase is the dual simulation phase that occurs before superstep 6. The ball creation occurs between supersteps 6 and 19 for strong and strict simulation, while it finishes at superstep 13 for tight simulation. Supersteps 20 and 21 in strong and strict algorithms correspond to...
processing balls and terminating the algorithms. This phase occurs in supersteps 14 and 15 of tight simulation. It is clear that, there is no communication in these supersteps.

There is a difference in communication at the first superstep because in strong simulation every vertex needs to learn about its neighborhood regardless of its matching status. Strict and tight algorithms perform exactly the same in the first phase. The exponential increase of communication size in strong simulation during creating balls is because of the involvement of all vertices in that process. The jagged shape of communication is because of our BFS-style algorithm for discovering balls, which contains requests at one superstep and responses at the next. Expectedly, the communication size of tight simulation is less than strict during the second phase.

Figure 11.d displays the difference between the numbers of active vertices in the three different types of simulation. Although the number of balls is smaller in tight in comparison to strict simulation, it has the same number of active vertices in the phase of ball creation because all the vertices of dual-match result graph stay active in this phase.

After the dual simulation phase, though the number of active vertices in strict and tight simulation does not experience a significant change, the number of active vertices in strong simulation increases exponentially because inactive vertices become active again during ball creation.

7. Related Work

We surveyed related work in two categories: pattern matching models, and distributed algorithms for pattern matching on massive graphs.

Many matching models find similarity between graphs based on exact structural matching like subgraph isomorphism and its approximate algorithms. On the other hand, some matching models take into account the meaning of attributes of vertices and edges and their relationship in addition to graph structure. We are interested in the recent type which is also called semantic graph matching (Gallagher, 2006).

A few graph pattern matching models introduced during recent years to find a proper semantic graph match like p-homomorphism (Fan, Li, Ma, Wang, & Wu, 2011) and bounded simulation (Fan, Li, Ma, Tang, Wu, & Wu, 2010). Ma et al., in (Ma, Cao, Huai, & Wo, 2011), talk about the idea of a distributed algorithm in abstract, but it is not based on vertex-centric and thus varies greatly from our approach. There are also very interesting theoretical investigations of different types of simulation presented in (Fan, 2012), but it never comes close to a distributed algorithm for strong simulation.

In (Ma, Cao, Huai, & Wo, 2012) a distributed algorithm only for basic graph simulation is proposed. They have implemented and tested their algorithm on a cluster of 16 machines; however, neither the algorithm nor its implementation is adapted to a widely-used cluster computing platform, but it is implemented all in Python. To best of our knowledge, our algorithms are the first distributed algorithms for dual and strong simulation.

There are also some proposed approaches based on indexing to increase efficiency of answering graph pattern matching queries; for example (Jin & Yang, 2011) or (Cheng, Yu, Ding, Yu, & Wang, 2008). However, creating and storing indexes for massive data graphs may not be feasible.

In (Sun, Wang, Wang, Shao, & Li, 2012) a distributed algorithm is introduced for finding isomorphic subgraph matches in a huge data graph. They deploy graphs on Trinity (Shao, Wang, & Li, 2012) which is a commercial distributed memory cloud environment. To avoid the construction time and the storage capacity for complicated indices, they only use a simple string index which maps node labels to node IDs. Conceptually similar to our approach, their main idea is based on graph exploration, but they cannot avoid expensive joins because of the intrinsic limits of subgraph isomorphism.

Our new simulation model, tight simulation, is a novel improvement of strict simulation which was introduced in (Fard, Nisar, Ramaswamy, Miller, & Saltz, 2013). Its distributed algorithm is also a modification of the distributed algorithm for strict simulation. The experiments for other distributed algorithms are complementary to the previous experiments presented in (Nisar, Fard, & Miller, 2013).

8. Conclusions

This paper discusses a spectrum of graph pattern matching models, and compares them in terms of the quality of the result and scalability. Also, a new model, named tight simulation, is introduced which is more scalable than the previous models in its family and yields higher quality results. Moreover, novel vertex-centric distributed algorithms for five simulation models are presented. The implementation of the distributed algorithms follow a vertex-centric BSP approach which has been successful in other areas of graph analytics, but has not yet been completely explored in the area of graph pattern matching. The detailed experimental study shows the benefits and costs of the proposed distributed algorithms for various graph simulation models. Our new technique for pattern matching, tight simulation, has been shown to scale better than previous simulation models in a vertex-centric framework.

In the future, we will explore the efficiency of similar techniques for graph pattern matching when the edges of the query and data graph are also labeled. We will also adapt algorithms to use incremental graph pattern matching techniques.

9. Acknowledgment

The authors would like to thank Semih Salihoglu, the developer of GPS from Stanford University, for his valuable
advice regarding the installation and usage of GPS. We would like also to thank Yang Cao, one of the authors of (Ma, Cao, Huai, & Wo, 2012), for sending us his Python code about distributed graph simulation. We are also grateful to Ayushi Jain for her programming efforts.

10. REFERENCES


Proceedings of the 28th ACM symposium on Principles of distributed computing (pp. 135-146). ACM.


Authors

Arash Fard is a PhD candidate in the Department of Computer Science at The University of Georgia. His research interests include distributed computing and big data analytics, especially for graph algorithms. He received a Khwarizmi International Award in 2008 for applied research on Cluster Computing.

M. Usman Nisar recently received his M.S. in Computer Science from the University of Georgia. He is interested in big data analytics and distributed computing, and planning to pursue his PhD in the same area.

John A. Miller is a Professor of Computer Science at the University of Georgia. His research interests include database systems, simulation, Web services and bioinformatics. Dr. Miller received the B.S. degree in Applied Mathematics from Northwestern University in 1980 and the M.S. and Ph.D. in Information and Computer Science from the Georgia Institute of Technology in 1982 and 1986, respectively. During his undergraduate education, he worked as a programmer at the Princeton Plasma Physics Laboratory. He is has been an Associate Editor for the ACM TOMACS, IEEE TSMC and SCS SIMULATION as well as an Editorial Board Member for the JOS and International Journal of SPM.

Dr. Lakshmish Ramaswamy is an Associate Professor in the Dept. of Computer Science at the University of
Georgia. There, he directs the Data Intensive Pervasive Systems Lab. Dr. Ramaswamy’s research interests include Data Intensive Systems, Pervasive Systems, and Distributed Computing. He has published extensively in these areas. Dr. Ramaswamy has received the best paper awards at the CoopIS-2010 and WWW-2004 conferences and the Pat Goldberg best paper award in 2005. He obtained a PhD from the Georgia Institute of Technology. His research is supported by grants from the National Science Foundation. Dr. Ramaswamy is a member of the IEEE.
A Throughput Driven Task Scheduler for Batch Jobs in Shared MapReduce Environments

Xite Wang, Derong Shen, Ge Yu, Tiezheng Nie, Yue Kou
College of Information Science and Engineering, Northeastern University, P.R. China
wangxite@research.neu.edu.cn, {shenderong, yuge, nietiezheng, kouyue}@ise.neu.edu.cn

Abstract
MapReduce is one of the most popular parallel data processing systems, and it has been widely used in many fields. As one of the most important techniques in MapReduce, task scheduling strategy is directly related to the system performance. However, in multi-user shared MapReduce environments, the existing task scheduling algorithms cannot provide high system throughput when processing batch jobs. Therefore, in this paper, a novel scheduling technique, Throughput-Driven task scheduling algorithm (TD scheduler) is proposed. Firstly, based on the characteristics of shared MapReduce environments, we propose the framework of TD scheduler. Secondly, we classify the jobs into six states. Jobs in different states have different scheduling priorities. We also give the rules of state conversion, which can ensure the fairness of resource allocation and avoid wasting system resources. Thirdly, we design the detailed strategies for job selection and task assignment. The strategies can effectively improve the ratio of local task assignment and avoid hotspots. Fourthly, we show that our TD scheduler can be applied to the heterogeneous MapReduce cluster with small modifications. Finally, the performance of TD scheduler is verified through plenty of simulation experiments. The experimental results show that our proposed TD scheduler can effectively improve the system throughput for batch jobs in shared MapReduce environments.

Keywords: MapReduce; shared environment; batch job; task scheduling; throughput

1. INTRODUCTION
Nowadays, various of applications, such as internet access, business computing and scientific research, generate large amount of data continually. Thus, the analysis of big data becomes a very hot topic which attracts the attention of many researchers\(^{[12]}\). As a popular parallel data processing system, MapReduce\(^{[3]}\) has been proven to be an efficient technique for big-data analysis, and it has been adopted in many fields. With the help of the MapReduce system, users can expediently process PB level of data without considering the execution details (e.g. data distribution, fault-tolerance).

Each MapReduce job contains several map and reduce tasks. Based on a specific scheduling strategy, the system will assign the tasks to the corresponding computing nodes and process them in parallel. Obviously, task scheduling is quite crucial for the system performance. An unbefitting task scheduler may cause lots of network transmission and reduce the system processing capacity.

In many practical applications, a large scale MapReduce cluster is often shared by several users. In a period of time, the cluster may receive a batch of jobs submitted by different users. Each job applies for a percentage of computing resources based its workload and the degree of importance, and all the jobs must be finished within the required time. Hence, in this shared MapReduce environment, the system needs a suitable scheduling strategy that can provide high throughput along with the fairness of resource allocation. In fact, the above problem is quite common. For example, an online shopping company (e.g. Amazon) builds a large MapReduce cluster shared by many departments. In the period that the internet access is rare (e.g. midnight), these departments will submit many MapReduce jobs, such as access log analysis, transaction information collation etc. Each job will apply a proportion of computing resources, and the system must finish all these jobs within this period so that the daily work is not affected. In this case, the system must first improve the throughput to finish the jobs on time. Meanwhile, the fairness of resource allocation should also be considered to avoid that individual job monopolizes the system resources.

In order to solve the problem above, in this paper, we propose TD scheduler which is designed for improving the system throughput for batch jobs in a shared MapReduce environment. In many MapReduce applications, most of the calculations are processed in the map phase, and the map phase determines the time cost of the entire job. Thus, our TD scheduler is mainly designed for the map phase. The contributions of this paper are summarized as follows.

1. We state the task scheduling problem for batch jobs in a shared MapReduce environment, and propose the frame of TD scheduler.
2. We classify the jobs into 6 states and give the rules of state conversion, which can ensure the fairness of resource allocation and avoid the waste of computing resources.
3. We design the detailed strategies for job selection and task assignment, which can significantly reduce the network overhead and improve the system throughput.
4. Through a small amount of modifications, we show that our TD scheduler can be applied in the heterogeneous MapReduce cluster.
5. We verify the effectiveness of TD scheduler through plenty of experiments.

The rest of this paper is organized as follows. We overview the MapReduce framework and state several previous approaches of task scheduling as background knowledge in Section 2. We define the problem model and state the design objective in Section 3. We describe the TD scheduler in detail in Section 4. We extend the TD scheduler to the heterogeneous environment in Section 5. We evaluate the performance of TD scheduler in Section 6. Finally, the conclusion of this paper is stated in Section 7.

2. BACKGROUND KNOWLEDGE

In this section, we firstly overview the MapReduce framework. Then we review the related work.

2.1 MapReduce Overview

The MapReduce system is presented by Google in 2004. It is designed for the large-scale data processing in a cluster. In general, the input files are stored in a distributed file system (DFS), and the large files will be segmented into several splits in the same size. When a job arrives, the master node of MapReduce will split the job into a number of map and reduce tasks, and assign them to the idle slave nodes. Each slave node has several slots for processing tasks assigned by the master.

\[ \text{Input} \xrightarrow{*} \text{Map} \xrightarrow{\text{Shuffle}} \text{Reduce} \xrightarrow{\text{Output}} \]

Figure 1 describes the process of MapReduce. Once a MapReduce job is submitted, the system will split $j$ into $m$ map tasks and $r$ reduce tasks. Usually, $m$ is the split number of the input file of $j$, and $r$ is a default number or user specified. In the map phase, each mapper reads a specified split of the input file and converts it to a sequence key-value pairs of type $(k_1, v_1)$. For each pair, the mapper calls the map function and outputs a new pair of type $(k_2, v_2)$ as intermediate result. Then according to the partition function, all the intermediate pairs with the same key are grouped into a sequence and sent to the corresponding reducer. In the reduce phase, each reducer firstly merges the received pairs and generates new pairs of type $(k_3, \text{list}(v_2))$, then calls the reduce function and outputs the final results.

The task assignment of MapReduce follows the data locality principle. When a slot requires task, the local task will be allocated preferentially. If no local task exists, the scheduler choose a task which is close to the slot in the network in order to induce the network overhead.

2.2 Related Work

As the kernel technique of parallel data processing, task scheduling has been studied for years. There have been many excellent task schedulers in MapReduce\cite{5,8}.

The default task scheduler of the Hadoop system\cite{9}, Job Queue Task scheduler (also called FIFO scheduler) can provide a fundamental and effective method for task assignment. It sorts the jobs in the order of the submission time. Each time only one job can be scheduled in the system, and the later jobs will not be scheduled until all the former jobs are finished.

Some schedulers are designed for multi users to safely share a large MapReduce cluster. The Capacity scheduler\cite{10} allocates computing resources to different users according to the configuration, which can avoid individual user monopolizes the system. It supports multiple job queues, and each queue is associated with a percentage of system resources. In each queue, the jobs are scheduled by FIFO. If some resources are idle, then will be shared with the other queues. Fair scheduler\cite{11} is also used for the cluster shared by multi users, and the design objective is "fairness". Specifically, it supports several job pools and each pool is constituted by some jobs with specific weights. Fair scheduler assign computing resources to each job according to its weight. The idle resources will be allocated to the job with the most vacancies. Based on Fair scheduler, Matei Zaharia et al. presented delay scheduling\cite{12,13} which can further improve the data locality. Furthermore, Thomas Sandholm et al.\cite{14} presented a scheduling strategy which supports dynamic resource allocation. The scheduler allows users to apply for more computing resources if the jobs they submit are important.

In practice, the performance of the nodes in a cluster is different, the computing power of some old machines may be much lower than the new ones. Therefore, Matei Zaharia et al. presented the LATE Scheduler\cite{15} for the heterogeneous environment. The scheduler estimates the processing ability of each node based on the accomplished tasks. If a task is executed slower than the others, it will be restarted on the node with good processing capacity. YongChul Kwon et al. discussed the skew problem in MapReduce and presented skewtune\cite{16}. The technique detects the skew by estimating the finish time of tasks. The rest of a slower tasks will be split and assigned to the other nodes to be executed in parallel.

Besides, several studies focus on task scheduling for MapReduce jobs with deadlines. Jorda Polo et al. presented PD scheduler\cite{17} applying to the deadline scheduling. Each job is associated with a specific deadline. According to the accomplished tasks, the scheduler calculates the expected finish time of the job. If a job cannot finish before the deadline with the allocated resources, the idle resources will be allocated to it preferentially. Kamal Kc et al.\cite{18} presented a job execution cost model to estimate the finish
time of a job. A job will be accepted if the expectant finish time meets the deadline. Otherwise, the job will be rejected.

In brief, the previous studies rarely consider the system throughput problem for batch jobs in shared MapReduce environments. In contrast, our TD scheduler can provide better throughput and has strong practicability.

3. Problem Statement

In this paper, we mainly focus on the system throughput problem in MapReduce. Firstly, the environment can be formalized as follows. For a MapReduce cluster with $T$ task slots, the jobs submitted by different users form a job queue $Q=\{j_1, j_2, \ldots, j_n\}$. For each job $j$ in $Q$, the related input file is stored in DFS, and $j$ is associated with a parameter $j.demand$ which means the number of task slots that $j$ expects to occupy. $j.demand$ can be transformed to other common parameters (such as job weight, deadline, etc.) easily, and it has good generality.

In order to provide good throughput, the system adopt the threshold method to ensure the fairness of resource allocation. Given a user specified lower bound $L$ ($0<L<1$), the system guarantees the number of task slots that a running job $j$ occupies: $j.occupied \geq L \times j.demand$. Given a user specified upper bound $H$ ($H>1$), the system guarantees the number of task slots that a running job $j$ occupies: $j.occupied \leq H \times j.demand$, which can avoid individual job monopolizes the computing resources.

Thus, given a MapReduce cluster with $T$ slots and a job queue $Q$ with $n$ jobs, under the premise that all the running jobs satisfy the constraints of $L$ and $H$, the goal of our TD scheduler is to effectively improve the system throughput so that all the jobs in $Q$ can finish as quickly as possible.

4. TD Scheduler Description

In this section, we describe the details of TD scheduler. Firstly, we give the framework of TD scheduler. Then, we introduce the job states. At last, we describe the method of task assignment.

4.1 Frame of TD Scheduler

![Figure 2. Frame of TD Scheduler](image)

In general, there are 3 basic states for the jobs in $Q$. a) waiting, none of the task of the waiting job has been scheduled, and the set of the waiting jobs is denoted as $S_{wait}$ (for simplicity, we assume that all the jobs in $S_{wait}$ have been sorted in the order of submission time). b) running, only the running jobs need to be scheduled, and the set of the running jobs is denoted as $S_{run}$. c) completed, all the tasks of the completed job have been finished, and the set of the completed jobs is denoted as $S_{com}$.

Figure 2 describes the frame of TD scheduler. When a slot becomes idle and it requests a task, the scheduler needs to select a suitable task and assign it to the slot. Firstly, the scheduler consider the jobs in $S_{run}$ and choose a job (e.g. $j_2$). Then, an untreated task of this job will be selected by TD scheduler and assigned to the slot.

4.2 Job States

According to the frame of TD scheduler, only the jobs in $S_{run}$ need to be scheduled. However, in order to ensure the allocation fairness for the running jobs and make full use of computing resources, basic job classification is not enough. Hence, we introduce 4 new job states in this section.

In the initialization phase, all the jobs in $Q$ are in state waiting, $S_{wait}=Q$. All the $T$ slots are idle and running job set $S_{run}$ is empty. Intuitively, the first $k$ jobs of $S_{wait}$ can be inserted into $S_{run}$ if

$$\sum_{i=1}^{k} j.demand \leq T$$

(1)

Along with the processing, some of the jobs in $S_{run}$ are finished and inserted into the completed job set $S_{com}$. Then some jobs in state waiting join into $S_{run}$. The updating of $S_{run}$ follows the principle below.

PRINCIPLE 1 For $\forall j_i \in Q$ ($i>1$), $j_i$ can join into $S_{run}$ only if $j_{i-1} \in S_{run}$.

In order to ensure the allocation fairness and high utilization rate of computing resources, based on the main parameters of a job (including the number of slots $j$ occupies $j.occupied$, the number of slots $j$ expects $j.demand$, the number of untreated tasks of $j$ $j.remain$), we classify the running jobs into 4 types.

As Figure 3 shows, there are 4 states for the running jobs, including infantile, teenaged, adult and senile. The corresponding job sets are denoted as $S_{inf}$, $S_{teen}$, $S_{adult}$, $S_{sen}$. The jobs in different states have different scheduling priorities. The states convert only follows the arrows. Next, we describe the states in more detail.

![Figure 3. Job States](image)

Infantile is a specific job state, and for all the running jobs, at most 1 job is in state infantile. The purpose of state infantile is to keep that all the slots can obtain tasks and
avoid resource wasting. The job in state infantile is not restrained by the bounds and has low scheduling priority.

Theoretically, when the running jobs cannot occupy all the slots, some slots may get no task to run. In order to avoid resource wasting, the first job in $S_{wait}$ will join into $S_{inf}$ for high system utilization. Formally, a waiting job $j_{wait}$ can join into $S_{inf}$ if

$$D + \sum_{j=s} j_{occupied} < T$$

$$j_{wait}.demand > T - D - \sum_{j=s} j_{occupied}$$

where $D = \sum_{j=s} j.demand$.

**Teenaged** is the beginning state for a running job $j$, which means that the system already has enough slots for $j$ (more than $j$ expects), but actually the number of slots that $j$ has occupied does not meet the constraint of lower bound (less than $L \times j.demand$). In order to ensure the allocation fairness, the jobs in state teenaged have the highest priority.

With the jobs in $S_{run}$ finishing gradually, the system obtains more slots to reallocate. For a infantile job $j_i$, if the number of slots that the system can reallocate is more than $j_i.demand$, but the number of slots that $j_i$ occupies does not meet the constraint of lower bound, $j_i$ will be converted to a teenaged job. Formally, $j_i$ can join into $S_{teen}$ if

$$j_i.demand \leq T - D - \sum_{j=s} j_{occupied}$$

$$j_i.occupied < L \times j_i.demand$$

**Adult** is the general state for a running job. The number of slot that an adult job is stable and meets the constraints of both the lower bound and the upper bound. The jobs in state adult have the normal scheduling priority.

For an infantile job $j_i$, if the number of slots that the system can reallocate is more than $j_i.demand$, and the number of slots that $j_i$ occupies satisfies the constraints of lower bound and upper bound, $j_i$ will join into $S_{adult}$. Formally, $j_i$ can join into $S_{adult}$ if

$$j_i.demand \leq T - D - \sum_{j=s} j_{occupied}$$

$$L \times j_i.demand \leq j_i.occupied \leq H \times j_i.demand$$

If the number of slots that a teenaged job $j_t$ occupies reaches $L \times j_t.demand$, $j_t$ can be converted to an adult job. The conversion condition is similar to Equation 4, we do not give the unnecessary details.

**Senile** is last state for a running job. The job in state senile still has tasks in processing, but has no untreated task. The running tasks of a senile job consume some system resources, but do not need to be further scheduled or meet the constraint of lower bound.

When all the tasks of an adult job $j_a$ have been assigned to the slots, $j_a$ will join into the senile job set $S_{sen}$. The conversion condition is

$$j_a ремаіn = 0$$

$$j_a . occupied \neq 0$$

For a senile job $j_s$, along with the accomplishment of the running tasks, $j_s$ finishes and it will join into the completed job set $S_{com}$. When all the jobs in $Q$ join into $S_{com}$, the process completes.

Thus far, we have introduced all the job states and the conditions of state conversion. Next, Table 1 shows an example of job states.

As Table 1 describes, There is a MapReduce cluster with $T=1000$ slots, and a job queue $Q=\{j_1, j_2, j_3, j_4, j_5\}$, lower bound $L=0.7$, upper bound $H=1.3$. In the initialization phase, according to Equation 1, $j_1, j_2, j_3, j_4$ can join into the running job set $S_{run}$. At some point, there is a slot becomes idle and requests task to run, and the processing details are described in case 1. $j_1.ремаіn=0$ means that $j_1$ has no remaining untreated task and it is in state senile. $j_2, j_3, j_4$ are in state adult. Hence, according to Equation 2, the theoretical number of slots that running jobs can occupy is $(200+150+250)+399<1000$, and $j_5$ is converted to state infantile. Then after several times of task assignments, the system receives a task request. If the situation is like case 2(a), according to Equation 3, the theoretical number of slots that the system can allocate to $j_5$ is $(200+150+250)-200 \times 0.7$. Hence $j_5$ joins into the teenaged job set. If the situation is like case 2(b), according to Equation 4, $j_5.occupied=143 \times 200\times 0.7$. Hence $j_5$ is converted to state adult immediately.

**Table 1 Example of State Conversion**

<table>
<thead>
<tr>
<th>Case</th>
<th>Job parameters</th>
<th>$j_1$</th>
<th>$j_2$</th>
<th>$j_3$</th>
<th>$j_4$</th>
<th>$j_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$j.demand$</td>
<td>400</td>
<td>200</td>
<td>150</td>
<td>250</td>
<td>200</td>
</tr>
<tr>
<td>2(a)</td>
<td>$j.occupied$</td>
<td>399</td>
<td>212</td>
<td>142</td>
<td>246</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>$j.remain$</td>
<td>&gt;0</td>
<td>&gt;0</td>
<td>&gt;0</td>
<td>&gt;0</td>
<td>&gt;0</td>
</tr>
<tr>
<td></td>
<td>$j.state$</td>
<td>senile</td>
<td>adult</td>
<td>adult</td>
<td>adult</td>
<td>wait - infantile</td>
</tr>
<tr>
<td>2(b)</td>
<td>$j.occupied$</td>
<td>200</td>
<td>248</td>
<td>165</td>
<td>285</td>
<td>101</td>
</tr>
<tr>
<td></td>
<td>$j.remain$</td>
<td>&gt;0</td>
<td>&gt;0</td>
<td>&gt;0</td>
<td>&gt;0</td>
<td>&gt;0</td>
</tr>
<tr>
<td></td>
<td>$j.state$</td>
<td>senile</td>
<td>adult</td>
<td>adult</td>
<td>infantile - teenaged</td>
<td></td>
</tr>
</tbody>
</table>

### 3.2 Task Assignment

The task assignment contains 2 main steps. a) Select a suitable job from the running job set. b) Select a suitable task of the job. There are 2 types of assignments. a) Forced assignment is used to ensure the allocation fairness. b) Normal assignment is used for improving the system throughput. Next, we firstly introduce the 2 cases which trigger the forced assignment.

**CASE 1** When a slot finishes a task which belongs to job $j$ and requires a new task to run, if $j \in S_{adult} \cup S_{teen}$ and
\( j.\text{occupied} = j.\text{demand} - 1 \), we must assign a task of \( j \) to the slot in order to guarantee that \( j \) satisfies the constraint of the lower bound.

**CASE 2** If case 1 is not triggered but the teenaged job set \( S_{\text{teen}} \neq \emptyset \), we must assign a task of the first job in \( S_{\text{teen}} \) to the slot because the teenaged job has higher priority.

If the forced assignment is triggered, the specific job is selected. Then the task selection of this job follows the MapReduce default data locality principle (mentioned in Section 2.1). If the forced assignment is not triggered, the scheduler will assign the tasks normally, and the details are described as follows.

**A. How to select a job**

In order to maximize the system throughput, we mainly consider the following 2 principles for job selection.

**PRINCIPLE 2** High ratio of local processing can reduce the unnecessary network traffic.

In the parallel computing, data locality can significantly reduce the network overhead, and it is the most efficient way to improving the system throughput. However, the existing task schedulers for multi-user MapReduce clusters do not consider the data locality as the primary target. For example, the Fair scheduler assigns tasks mainly based on the job weight. Besides, the FIFO scheduler has considered the data locality, but it does not support multi-job co-scheduling and cannot be used in a shared cluster. In our TD scheduler, we preferentially select the job that has the local tasks, which can improve the throughput effectively.

We denote the average processing time per task of job \( j \) as \( j.\text{amt} \), and it can be estimated easily based on the completed tasks of \( j \). Then we give another principle.

**PRINCIPLE 3** For the local task assignment, the job which has smaller \( \text{amt} \) value has higher priority to be selected. For the nonlocal task assignment, we choose the job with large \( \text{amt} \) value.

For the local task assignment, the task of the job with small \( \text{amt} \) value can be completed within a short time, and the slot can be released earlier, which can improve the flexibility of scheduling. For the nonlocal task assignment, because MapReduce adopts the streaming process mode, if we select a task of the job with large \( \text{amt} \) value, the delay ratio\(^1\) is small, which can indirectly improve the throughput. In addition, more tasks of the jobs with small \( \text{amt} \) values can be left for local assignment.

The principle can be explained through the following example. For job \( j_1, j_2 \), the split size of the input file is 128MB, and \( j_1.\text{amt} = 60s, j_2.\text{amt} = 10s \). Hence if we choose a task of \( j_1 \) for nonlocal assignment, the delay ratio is small if the network bandwidth can reach 128/60=2.13MB/s. For \( j_2 \), the bandwidth needs to reach 128/10=12.8MB/s. However, the network resources are limited in most of the distributed computing. We assume network bandwidth is 2MB/s. If we choose a task of \( j_1 \) for nonlocal task assignment, the shortest processing time is 128/2=64s, and the delay ratio of \( j_1 \) is (64-60)/64=0.063. If we choose a task of \( j_2 \), the shortest processing time is 64s, and the delay ratio is (64-10)/64=0.844. Hence, for a nonlocal task assignment, the task of \( j_1 \) (the job with larger \( \text{amt} \) value) can reduce the processing delay and improve the system throughput.

When a slot on node \( n \) requires task to the master, according to the principles above, the method of job selection can be summarized as follows.

Step 1. We attempt to make a local assignment. Firstly, we sort all the jobs in \( S_{\text{adu}} \) according to the \( \text{amt} \) values and check them one by one. For a job \( j \), if \( j \) has untreated tasks stored on node \( n \) and \( j \) satisfies the constraint of the upper bound \( j.\text{occupied} < H \times j.\text{demand} \), \( j \) is selected and a local task of \( j \) will be allocated. If there is no satisfied job in \( S_{\text{adu}} \), we further check the infantile job. If the infantile job does not meet the condition, a nonlocal task assignment is unavoidable (step 2).

Step 2. If a nonlocal assignment is necessary, we sort the jobs in \( S_{\text{adu}} \) according to the \( \text{amt} \) values in descending order. For each job \( j \), if \( j \) satisfies the constraint of the upper bound, \( j \) will be selected. If on adult job satisfies the constraint, the infantile job will be selected. Next, we introduce the method of selecting a suitable task of a given job.

**B. How the select a task**

After we select a specific job \( j \), we have already known that whether the task assignment is local or not. If it is local, the method of task selection is straightforward and we do not give the unnecessary details. If we need to choose a nonlocal task, the network transmission is unavoidable. Basically, we intend to choose the task which is close to the slot in the network in order to reduce the network congestion. Furthermore, the hotspot problem is another enemy of the system throughput, and it cannot be ignored. Hence, we present a novel task selection method which can avoid hotspots.

In general, a hotspot is the node which transmits data with too many nodes simultaneously. In order to further analyze the problem, we classify the hotspots into two categories.

The node \( n \) is an actual hotspot if the total number of the nodes which transmit data with \( n \) is larger than the threshold \( \epsilon \). In other words, given the network bandwidth \( \varphi \), if a nonlocal task reads data from \( n \) across the network, the performance will be affected if the data transmission speed cannot reach to \( \varphi/\epsilon \).

To avoid actual hotspots, we employ a new data structure hot queue. a) Append operation. If there is a new nonlocal task assignment that a slot needs to read data from node \( n \) at time point \( t \), a tuple \((n, t)\) will be appended to the end of hot queue. b) Remove operation. We denote the split size of the input file as \( \gamma \) and the network transmission speed as \( \varphi/\epsilon \). Periodically, we check the tuples of hot queue.

\(^1\)For a task \( t \), we denote the time cost of \( t \) as \( c_t \); if \( t \) is processed locally, and \( c_t \) if \( t \) is processed non-locally. Then the delay ratio of \( t \) is \( (c_t-c_j)/c_t \).
in order. For each tuple \((n, t)\), if \(t + \varepsilon \leq \tau_{wor}\) where \(\tau_{wor}\) is the current time, the tuple is overdue and we remove it from the hot queue. In addition, there is a corresponding statistical table. For each node \(n\) in the hot queue, the table records \(\text{count}(n)\), the total number of the tuples which contain \(n\). If \(\text{count}(n) = \varepsilon\), we forbid more connections to \(n\) until some tuples which contain \(n\) become overdue.

Besides, node \(n\) is a potential hotspot if the remaining untreated workload on \(n\) is larger than that on the other nodes. Because in the foreseeable future, there is no local task on the other nodes, and lots of nodes need to read data from \(n\). The potential hotspot \(n\) becomes an actual hotspot.

To avoid potential hotspots, we firstly formalize the remaining untreated workload on node \(n\).

\[
R_n = \sum_{j=\text{task of } n} \text{amt} \times r^j_n
\]

where \(r^j_n\) is the number of the remaining untreated tasks of job \(j\) stored on node \(n\). Then the potential hotspot with the largest remaining workload has the highest probability to become an actual hotspot, and we intend to select the task on this node to reduce its remaining workload.

In summarize, if the scheduler needs to select a nonlocal task of job \(j\) for the idle slot, we firstly get the nodes which satisfy the following 2 conditions. a) The node still has local tasks of \(j\). b) The current connection number of the node is smaller than \(\varepsilon\) according to the statistical table. Then from these nodes, we choose the node which has the most remaining workload and assign a task of \(j\) stored on this node to the idle slot.

**Table 2 Example of Nonlocal Task Selection**

<table>
<thead>
<tr>
<th>node</th>
<th>connection number</th>
<th>number of tasks stored on node</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n_1)</td>
<td>0</td>
<td>(r^{t_{\text{inf}}}<em>n=4), (r^{t</em>{\text{teen}}}<em>n=5), (r^{t</em>{\text{adult}}}_n=0)</td>
</tr>
<tr>
<td>(n_2)</td>
<td>3</td>
<td>(r^{t_{\text{inf}}}<em>n=5), (r^{t</em>{\text{teen}}}<em>n=3), (r^{t</em>{\text{adult}}}_n=4)</td>
</tr>
<tr>
<td>(n_3)</td>
<td>0</td>
<td>(r^{t_{\text{inf}}}<em>n=3), (r^{t</em>{\text{teen}}}<em>n=0), (r^{t</em>{\text{adult}}}_n=7)</td>
</tr>
<tr>
<td>(n_4)</td>
<td>0</td>
<td>(r^{t_{\text{inf}}}<em>n=3), (r^{t</em>{\text{teen}}}<em>n=6), (r^{t</em>{\text{adult}}}_n=7)</td>
</tr>
<tr>
<td>(n_5)</td>
<td>1</td>
<td>(r^{t_{\text{inf}}}<em>n=6), (r^{t</em>{\text{teen}}}<em>n=4), (r^{t</em>{\text{adult}}}_n=6)</td>
</tr>
</tbody>
</table>

Table 2 shows an example of nonlocal task selection. The cluster contains 5 slave nodes, and job \(j_1,j_2,j_3\) are running in the system, \(j_1, \text{amt}=20s\), \(j_2, \text{amt}=25s\), \(j_3, \text{amt}=30s\). The connection threshold \(\varepsilon=3\). If an idle slot on node \(n_4\) requires a task, we select job \(j_1\) and have to make a nonlocal task assignment. Then, we firstly collect the nodes which still have untreated tasks of \(j_1\), \(\{n_2,n_3,n_5\}\). Because the connection number \(n_5\) is equal to \(\varepsilon\), we remove \(n_5\). Then we compute the remaining workload of \(n_5\), \(R_{\text{amt}}=270\), \(R_{\text{ew}}=400\). Thus \(n_5\) is selected, and a task of \(j_1\) on \(n_5\) is assigned to the idle slot.

### 3.3 Algorithm Description

In this section, we illustrate the TD scheduler in more detail. Algorithm 1 describes the frame of TD scheduler. When a slot on node \(n\) finish a task of job \(j\), it sends a task request to the master, and then TD scheduler is invoked. First, update the number of slots that \(j\) occupies, \(j, \text{occupied}\) - (line 1). If \(j\) is a teenaged or adult job and \(j, \text{occupied}\) \<\(\text{demand} \times L\), the forced assignment is triggered, and we assign a task of \(j\) to the slot (line 2, 3). Otherwise, if \(j\) is a senile job, we need to update the job state (line 6-17). In particular, if \(j, \text{occupied}=0\), \(j\) is completed and we move it into \(S_{\text{com}}\) (line 7). If there is a infantile job \(j\) and \(j, \text{demand} \leq T-D \Sigma_{r_\in S_{\text{run}}} j, \text{occupied}\), we need to move \(j\) into \(S_{\text{run}}\) or \(S_{\text{da}}\) based on Equation 3 and 4 (line 11-15). If there in no infantile job and \(\Sigma_{r_\in S_{\text{run}}} j, \text{occupied} + T < T\), we move the first waiting job into \(S_{\text{aw}}\) based on Equation 2 (line 16, 17). Then if \(S_{\text{run}}\) is not empty, the forced assignment is triggered, and we assign a task of the first teenaged job to the slot (line 18-20). Otherwise, we assign a task to the slot normally (line 21, 22). The function \(\text{AssignTask}_\text{Normal}(n)\) is introduced in Algorithm 2. At last, we update the parameters of the selected job and the job state (line 23-27).

**Algorithm 1 TD Scheduler**

**Input:** the node \(n\) which contains the idle slot, the job \(j\) which the task processed by the slot belongs to

**Output:** a task for the idle slot

01: \(j, \text{occupied}--\);  
02: if \((j \in S_{\text{run}}) \land j, \text{occupied} \leq \text{demand} \times L)\)  
03: \(j_{\text{ass}} \leftarrow j\);  
04: \(\text{AssignTask}_\text{SpecificJob}(j_{\text{ass}}, n)\);  
05: else  
06: if \((j \in S_{\text{run}})\)  
07: if \((j, \text{occupied} = 0)\)  
08: \(j\) joins into \(S_{\text{com}}\);  
09: if \((S_{\text{inf}} \neq \emptyset)\)  
10: get the infantile job \(j\) from \(S_{\text{inf}}\);  
11: if \((j, \text{demand} \leq T - \Sigma_{r_\in S_{\text{run}}} j, \text{occupied})\)  
12: if \((j, \text{occupied} < j, \text{demand} \times L)\)  
13: \(j\) joins into \(S_{\text{run}}\);  
14: else  
15: \(j\) joins into \(S_{\text{da}}\);  
16: else if \((\Sigma_{r_\in S_{\text{run}}} j, \text{occupied} + D < T)\)  
17: move the first job in \(S_{\text{wait}}\) to \(S_{\text{inf}}\);  
18: if \((S_{\text{run}} = \emptyset)\)  
19: \(j_{\text{ass}} \leftarrow \) the first job in \(S_{\text{run}}\);  
20: \(\text{AssignTask}_\text{SpecificJob}(j_{\text{ass}}, n)\);  
21: else  
22: \(j_{\text{ass}} \leftarrow \text{AssignTask}_\text{Normal}(n)\);  
23: \(j_{\text{ass}}, \text{occupied} +, j_{\text{ass}}, \text{remain} \leftarrow -\);  
24: if \((j_{\text{ass}} \in S_{\text{run}}) \land j_{\text{occupied}} \geq j, \text{demand} \times L)\)  
25: \(j_{\text{ass}}\) joins into \(S_{\text{da}}\);  
26: if \((j_{\text{ass}}, \text{remain} = 0)\)  
27: \(j_{\text{ass}}\) joins into \(S_{\text{run}}\);  
28: end;

Algorithm 2 describes the function \(\text{AssignTask}_\text{Normal}(n)\), and the job selected by the function will be returned. Firstly, we sort all the jobs in \(S_{\text{run}}\) according to the \text{amt} values (line 1). Then we check the jobs in \(S_{\text{run}}\) in order. For each job \(j\), if it has untreated tasks stored on node \(n\) and satisfies the constraint of the upper bound, we select \(j\) and the algorithm ends (line 2-5). If on job is selected, we check the
infantile job (if there is) with the same condition (line 6-9). If on job satisfies, we check the jobs in $S_{ada}$ in reverse order. For each job $j$, if it meets the constant of the upper bound, we select $j$ and select a node by using the function GetNonHotspotNode($j$) (will be described in Algorithm 3) (line 10-14). If no job meets the conditions, in order to avoid wasting resources, we assign a task of the infantile job to the slot (line 15-17).

Algorithm 2 AssignTask_Normal ($n$)

Input: node $n$ which the idle slot belongs to
Output: assign a task of $j$ to the slot and return job $j$
01: sort the jobs in $S_{ada}$ according to the $amt$ values;
02: for each $j$ in $S_{ada}$
03: if ($n$ has untreated tasks of $j$ and $j.occupied < j.demand \times H$)
04: assign a local task of $j$ on $n$ to the slot;
05: return $j$;
06: get the infantile job $j$, from $S_{inf}$;
07: if ($n$ has untreated tasks of $j$ and $j.occupied < j.demand \times H$)
08: assign a local task of $j$ on $n$ to the slot;
09: return $j$;
10: for each $j$ in $S_{ada}$ in reverse order
11: if ($j.occupied < j.demand \times H$)
12: $n_{ass} \leftarrow$ GetNonHotspotNode($j$);
13: assign a task of $j$ on $n_{ass}$ to the slot;
14: return $j$;
15: $n_{ass} \leftarrow$ GetNonHotspotNode($j$);
16: assign a task of $j$, on $n_{ass}$ to the slot;
17: return $j$;

Algorithm 3 GetNonHotspotNode ($j$);

Input: the specific job $j$
Output: return a node which is not hotspot
01: job set $S_{get}$– get all the jobs which has untreated tasks of $j$;
02: remove the jobs whose current connection numbers are $e$ from these jobs (line 2). At last, we select the job from the remaining jobs according to Equation 6 and return (line 3,4).

5. Extending TD Scheduler for Heterogeneous Environments

In most of the researches, the authors assume that the MapReduce cluster is isomorphous for simplicity. However, in the real-world usage, it cannot be guaranteed that all the nodes in the cluster has the same computing capacity. In fact, there may be several generations of machines in a cluster, and the performance of these computers is different. In other words, the heterogeneous environment is very common. Hence, in this section, we extend our TD scheduler for heterogeneous environments.

Firstly, we use a test method to quantify the computing capacity of a node in the heterogeneous cluster. In detail, we can run the same program with the same input on each node and record the processing time. The length of the processing time is a good measurement for the computing capacity. We denote the processing time on node $n_i$ as $t_i$, and average processing time as $\overline{t}$. Then the capacity factor of node $n_i$, $n_i.cf = \frac{t_i}{\overline{t}}$, and we use capacity factor to measure the computing capacity of a node. Node with larger value of capacity factor has better computing capacity.

Table 3 Example of Capacity Factor

<table>
<thead>
<tr>
<th>Number</th>
<th>Processing Time (s)</th>
<th>Capacity Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_1$</td>
<td>10</td>
<td>1.3</td>
</tr>
<tr>
<td>$n_2$</td>
<td>12</td>
<td>1.08</td>
</tr>
<tr>
<td>$n_3$</td>
<td>15</td>
<td>0.87</td>
</tr>
<tr>
<td>$n_4$</td>
<td>10</td>
<td>1.3</td>
</tr>
<tr>
<td>$n_5$</td>
<td>18</td>
<td>0.72</td>
</tr>
</tbody>
</table>

For example in Table 3, there are 5 nodes in the cluster. We process the same program on each node and record the corresponding processing time. Then the capacity factors can be calculated easily. As Table 3 shows, node $n_5$ has the smallest capacity factor, which means it has the worst performance in the cluster.

By using the capacity factor, we modify the TD scheduler for heterogeneous environments. For each running job $j$, we denote the set of slots occupied by $j$ as $S_j$. For each slot $s$ in $S_j$, the node which $s$ belongs to is denoted as NodeOf($s$). Then, $j.occupied$ can be rewritten as

$$j.occupied = \sum_{s \in S_j} \text{NodeOf}(s).cf$$

(7)

Besides, we need to formalize the remaining workload of a node in a heterogeneous cluster. Then, Equation 6 is rewritten as

$$R_n = n.cf \times \sum_{j:s_{\text{occupied}} \in S_{\text{ada}}} j.amt \times r_{t_i}$$

(8)

With the simple modifications above, TD scheduler can be extended to heterogeneous environments easily.

6. Experimental Evaluation

In this section, we evaluate the performance of TD scheduler. The MapReduce cluster for experiments is constituted by Hadoop (version 1.0.4) with 1 master and 50 slave nodes. Each node has a Core i3 2100 @ 3.1GHz CPU, 8GB memory, 500GB hard disk. The operating system is Red Hat Linux 6.1. There are 2 map task slots and 2 reduce task slots on each node, and the size of the input split is 64MB. The default value of the lower bound $L=0.7$, upper bound $H=1.3$. 
we use the following 4 types of MapReduce jobs to constitute the job queue for testing.

Word count is the basic MapReduce job. For each word \( w \), the map function output a pair of type \((w, 1)\). The reduce function sums the "1"s of a given word, and outputs a pair of type \((w, \text{counter})\).

Inverted index is used for building index for a large file. The map function parses the input file. For each word \( w \), it outputs a pair \((w, \text{position of } w)\). The reduce function merges the positions of a given word and outputs pairs of type \((w, \text{list(position)})\).

Distributed grep is used for finding a give pattern from a large file. The map function outputs an intermediate result if it matches the pattern. The reduce function just copies the intermediates and outputs them.

Distributed sort is used for sorting large number of items in parallel, and it adopts continuous partition function. The map function just scans the items and sends them to the corresponding reducers based the partition function. The reduce function sorts the received items locally.

For each job \( j \), \( j.demand \in [15-40] \), and it is related to the job size (the split number of input file).

In order to imitate different real applications, we also generate 3 types of job queues. a) Small type means the size of each job in the queue is small (<100). b) Large type means each job size is large (>200). c) Mix type means the job size can be varied among [50 - 250]. The total input file size of the jobs in a queue spans [100 - 300] GB, the default value is 200GB.

We evaluation the performance of TD scheduler by comparing with the common FIFO scheduler and Fair scheduler. Firstly, we test the runtime of job queues, ratio of local task assignment and the occurrence number of hotspots. Then, we show the influence of the threshold. At last, we test the performance of TD scheduler in heterogeneous environments.

6.1 Result Analysis

(a) Runtime vs. Lower Bound \((H=1.3)\)  
(b) Runtime vs. Upper Bound \((L=0.7)\)

Figure 6. The Influence of Threshold
In Figure 4, we test the runtime of the job queues using different schedulers. For each type of job queue, the runtime of our TD scheduler is significantly shorter than that of the others, which means TD scheduler can provide better throughput. Although the FIFO scheduler shows better performance than Fair scheduler (close to TD), it is not designed for multi users sharing a cluster. Besides, by comparing the 3 subfigures in Figure 4, more small jobs in a job queue leads to more time consumption.

In Figure 5, We test the network overhead through calculating the ratio of local task assignment. For the large type, Fair and FIFO maintains high ratio of local assignment (about 70%). With more small jobs in a queue, the local ratio decreases. For the small type, the ratio is reduced to nearly 35%. Whereas, TD scheduler keeps a high ratio of local assignment to reduce the network overhead, which is the main reason that TD can provide good throughput.

We also test the occurrence number of hotspots. The whole process is divided into 5 stages. We repeat the same process 3 times and record the average numbers of hotspots in each stage. The result is showed in Table 4. Comparing the 3 subfigures in Figure 4, more small jobs in a queue, the local ratio decreases. For the small type, the ratio is reduced to nearly 35%. Whereas, TD scheduler keeps a high ratio of local assignment to reduce the network overhead, which is the main reason that TD can provide good throughput.

<table>
<thead>
<tr>
<th>scheduler</th>
<th>queue type</th>
<th>stage in processing (percent)</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>TD</td>
<td>small</td>
<td>1-20</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td></td>
<td>21-40</td>
<td>20.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>41-60</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td></td>
<td>61-80</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td></td>
<td>81-100</td>
<td>28.7</td>
</tr>
<tr>
<td></td>
<td>mix</td>
<td>1-20</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td></td>
<td>21-40</td>
<td>15.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>41-60</td>
<td>18.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>61-80</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>81-100</td>
<td>20.3</td>
</tr>
<tr>
<td></td>
<td>large</td>
<td>1-20</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>21-40</td>
<td>6.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>41-60</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>61-80</td>
<td>4.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>81-100</td>
<td>9</td>
</tr>
<tr>
<td>FIFO</td>
<td>small</td>
<td>1-20</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td></td>
<td>21-40</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td></td>
<td>41-60</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td></td>
<td>61-80</td>
<td>22.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>81-100</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>mix</td>
<td>1-20</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>21-40</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>41-60</td>
<td>15.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>61-80</td>
<td>16.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>81-100</td>
<td>19.3</td>
</tr>
<tr>
<td></td>
<td>large</td>
<td>1-20</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>21-40</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>41-60</td>
<td>3.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>61-80</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>81-100</td>
<td>6</td>
</tr>
<tr>
<td>TD</td>
<td>small</td>
<td>1-20</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>21-40</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>41-60</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>61-80</td>
<td>0.7</td>
</tr>
<tr>
<td></td>
<td>mix</td>
<td>1-20</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>21-40</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>41-60</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>61-80</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>large</td>
<td>1-20</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>21-40</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>41-60</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>61-80</td>
<td>0.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>81-100</td>
<td>0.7</td>
</tr>
</tbody>
</table>

Figure 6(a) describes the influence of lower bound. When $L > 0.7$, with the increase of $L$, the runtime gets longer, which means that the throughput gets lower. This is mainly because higher $L$ leads to tighter constraint of jobs, so more forced assignments happen when slots requires tasks. When $L < 0.7$, the constraint of jobs is loose enough, so the runtime gets stable. But too small $L$ may results in that some jobs are processed for quite a long time, which is not good for the user experience and the system flexibility. Hence the optimal value of the lower bound is 0.7. Similarly, we get the optimal value of the upper bound $H = 1.3$.

6.2 Evaluation in a Heterogeneous Cluster

In order to build a heterogeneous cluster, we classify the 50 slave nodes into 3 groups and lock the CPU main frequency of the nodes in each group. The detail configuration is shown in Table 5.

<table>
<thead>
<tr>
<th>group ID</th>
<th>number of nodes</th>
<th>CPU frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16</td>
<td>2GHz</td>
</tr>
<tr>
<td>2</td>
<td>17</td>
<td>2.5GHz</td>
</tr>
<tr>
<td>3</td>
<td>17</td>
<td>3.1GHz</td>
</tr>
</tbody>
</table>

Figure 7 shows the runtime in a heterogeneous cluster. Comparing with the results in Figure 4, we can see that the runtime gets longer because we reduce the CPU frequencies of some nodes. TD scheduler still shows better performance than FIFO and Fair, and it shows stable performance even in a heterogeneous cluster.

Through the experiments above, we verify the effectiveness of TD scheduler and show that our proposed scheduler can improve the system throughput significantly.

7. CONCLUSIONS

This paper focuses on the system throughput problem for batch jobs in a shared MapReduce cluster, and proposes a novel task scheduling algorithm, called TD scheduler. First, according to the difference of the job parameters, we classify the jobs into 6 states and design the rules of state conversion, which can ensure the fairness of resource allocation and avoid the waste of computing resources. We design the detailed strategies for job selection and task selection. The strategies improve the ratio of local task assignment and avoid the occurrence of hotspots, which can significantly improve the system throughput. We also show that our TD scheduler can be applied in the heterogeneous MapReduce cluster through a small amount of modifications. At last, we verify the effectiveness and practicability of TD scheduler through plenty of experiments.
8. ACKNOWLEDGMENT

This work is supported by the National Basic Research 973 Program of China under Grant No.2012CB316201, the National Natural Science Foundation of China under Grant Nos. 61033007, 61003060, the Fundamental Research Funds for the Central Universities under Grant No. N100704001, the National Research Foundation for the Doctoral Program of Higher Education of China Grant No. 2012042110028 and the MOE-Intel Special Fund of Information Technology under Grant No. MOE-INTEL-2012-06.

9. REFERENCES

http://hadoop.apache.org/docs/r0.20.2/capacity_scheduler.html
http://hadoop.apache.org/docs/r0.20.2/fair_scheduler.html

8. ACKNOWLEDGMENT

This work is supported by the National Basic Research 973 Program of China under Grant No.2012CB316201, the National Natural Science Foundation of China under Grant Nos. 61033007, 61003060, the Fundamental Research Funds for the Central Universities under Grant No. N100704001, the National Research Foundation for the Doctoral Program of Higher Education of China Grant No. 2012042110028 and the MOE-Intel Special Fund of Information Technology under Grant No. MOE-INTEL-2012-06.

9. REFERENCES

http://hadoop.apache.org/docs/r0.20.2/capacity_scheduler.html
http://hadoop.apache.org/docs/r0.20.2/fair_scheduler.html


Authors

Xite Wang, born in 1987, Ph. D. student. His research area is big-data management.

Derong Shen, born in 1964, Ph. D., professor, Ph. D. supervisor. Her research area includes entity search and distributed computing.

Ge Yu, born in 1962, Ph. D., professor, Ph. D. supervisor. His research area includes database and big-data management.

Tiezheng Nie, born in 1980, Ph. D., associate professor. His research area includes data quality, data integration.

Yue Kou, born in 1980, Ph. D., associate professor. Her research area includes entity resolution, web data management.
Abstract—Personalization is the process of adapting the output of a system to a user’s context and profile. User information such as geographical location, academic and professional background, membership in groups, interests, preferences, opinions, etc. may be used in the process. Big data analysis techniques enable collecting accurate and rich information for user profiles in particular due to their ability to process unstructured as well as structured information in high volumes from multiple sources. Accurate and rich user profiles are important for personalization. For example, such data are required for recommender systems, which try to predict elements that a user has not yet considered. However, the information used for personalization can often be considered private, which raises privacy issues. In this paper, we discuss personalization with big data analysis techniques and the associated privacy challenges. We illustrate these aspects through the ongoing EEXCESS project. We provide a concrete example of a personalization service, proposed as part of the project, that relies on big data analysis techniques.

Keywords—personalization, privacy, big data, user profiling, recommender systems, EEXCESS

I. INTRODUCTION

Personalization consists of adapting outputs to a particular context and user. It may rely on user profile attributes such as the geographical location, academic and professional background, membership in groups, interests, preferences, opinions, etc. Personalization is used by a variety of web based services for different purposes. A common form of personalization is the recommendation of items, elements or general information that a user has not yet considered but may find useful.

General purpose social networks such as Facebook.com use personalization techniques to find potential friends based on the existing relationships and group memberships of the user. Professional social networks such as LinkedIn.com exploit the skills and professional background information available in a user profile to recommend potential employees. Search engines such as Google.com use the history of user searches to personalize the current searches of the user.

Big data analysis techniques are a collection of various techniques that can be used to discover knowledge in high volume, highly dynamic, and highly varied data. Big data techniques offer opportunities for personalization that can result in the collection of very comprehensive user profiles. Big data analysis techniques have two strengths in particular that enable collecting accurate and rich information for personalization: (1) Big data analysis techniques process unstructured data as well as structured data. Unstructured data of different varieties generated by users is growing in volume with high velocity and contains lots of useful information about the users. (2) Big data analysis techniques can process high volume data from multiple sources. This enables linking user attribute data from different sources and aggregating them into a single user profile. Moreover, user information from different sources can be correlated to validate or invalidate the information discovered from one source.

On one hand, user profiling with big data techniques is advantageous for providing better services as we have discussed above. On the other hand, user profiling poses a significant threat to user privacy. One can assume that an ethical and trustworthy service would use the information collected for personalization purposes with the user’s explicit consent and only for the benefit of the user. However, services that are less inclined toward protecting user privacy, may use personalization data for a number of purposes which may not be approved by the user and which may result in loss of private information. One example is the utilization of personalization data for targeted advertising [1]. Another example is the selling of private information in user profiles to third parties for a profit. The third parties may then use the private information for commercial or even malicious purposes [2]. Privacy breaches may occur even when a service is willing to protect a user’s privacy [3].

The ongoing EEXCESS project aims to personalize user recommendations by making intensive use of user profiling and therefore collecting detailed information about users. The EEXCESS project has to address various privacy challenges which appear mainly due to the use of big data and related technologies. One of the major challenges is that the EEXCESS architecture is based on a federated recommender system in which future partners may join. The trustworthiness and the intent of these partners are not necessarily known. The information collected and disclosed to recommenders may not, in itself, be sensitive, however, cross-referencing it with external big data sources and analyzing it through big data techniques may create breaches in user privacy. Since, untrustworthy partners may have access to such big data sources and techniques, privacy becomes a clear challenge.

In this position paper, we highlight some of the privacy issues related to personalization using big data techniques. We illustrate them by using the ongoing EEXCESS research project as a use case, and describe work done towards addressing the associated privacy challenges in that context. We present the proposed EEXCESS architecture, the privacy goals,
and the approaches being considered to achieve those goals.

This paper is organized as follows. Section II discusses the objectives behind personalization and how it can be obtained through different big data techniques. Section III discusses privacy issues which may appear in systems relying on information about users and in particular personalization systems. Section IV describes the EEXCESS project, its goal of providing personalized content recommendation and the impacts considering privacy. Section V recalls the role of the user in relationship to personalization and privacy. Section VI gives an overview of the current state of reconciling privacy and personalization, illustrated through the planned requirements for the EEXCESS system. We conclude in Section VII.

II. PERSONALIZATION

A. What is personalization?

The ultimate goal of personalization is to provide the most adapted response to a user’s current need with the fewest explicit information provided by him/her. Many existing systems provide some form of personalization. Google search personalizes search results using information such as the user’s geo-location, IP address, search history and result click-thru. Facebook provides “friend” recommendations based on a user’s social network already known by the service. Many location-based services, at a very minimum, use a user’s geo-location to provide results near the users current position. Personalized advertisements and marketing solutions attempt to better understand buying habits in order to propose advertisements to users for products they could likely be interested in.

Personalization is not limited to online services. For example, medical analysis systems try to build patient profiles which are as fine-grained as possible (e.g., taking into account genetic information) in order to propose the most adapted treatment to the patient. Personalization even reaches industrial processes, e.g., the industrial process of printing. Many printing firms offer the possibility to personalize statement documents such as bank statements, with adapted advertisements and offers. With the arrival of technologies such as 3D printers, it is very likely that the near future makes even more room for personalization.

There are clear advantages to personalization. A typical example is the utilization of user profile data for targeted advertising [1]. This way users only receive advertisements that they have the most interest in and are not overwhelmed with advertisements for products they wouldn’t even consider buying. Another example is filtering out spam emails. Personalization also improves the impact of a given service. In search systems, it allows users to more quickly find the information they are looking for. More generally it relieves users from the information overload they face every day by letting the systems dig through the massive amounts of data on their behalf and letting them find the relevant data for the users.

B. How big data techniques allow for personalization?

Big data techniques are a collection of various techniques labeled as such that can be used to discover knowledge in high volume, highly dynamic, and highly varied data. Big data techniques offer opportunities for personalization that can result in very comprehensive user profiles. Big data techniques have two strengths in particular that enable collecting accurate and rich information for user profiles: (1) Big data techniques process unstructured data as well as structured data. Unstructured data of different varieties generated by users is growing in volume with high velocity and contains lots of useful information about the users. (2) Big data techniques can process high volume data from multiple sources. This enables linking user attribute data from different sources and aggregating them into a single user profile. Moreover, user information from different sources can be correlated to validate or invalidate the information discovered from one source.

We list some of the big data analyses techniques below that can be used for collecting information about a user and building a user profile. An extended list of big data techniques that can be used for personalization can be found in [4].

Network analysis. Network analysis algorithms are used to discover relationships between the nodes in a graph or a network. Network analysis is particularly useful in the context of social networks where important information about the user such as his friends, co-workers, relatives, etc. can be discovered. Social network analysis can also reveal central users in the network, i.e., users who exert the most influence over other users. This information can be used to populate the attributes of social and environmental contexts, individual characteristics, etc. in a user profile.

Sentiment analysis. Sentiment analysis is a natural language processing technique that aims to determine the opinion and subjectivity of reviewers. The Internet is replete with reviews, comments and ratings due to the growing popularity of web sites such as Amazon.com, Ebay.com, and Epinion.com where users provide their opinion on others users and items. Moreover, micro-blogging sites such as Twitter.com and social network sites such as Facebook.com also hold a large amount of user opinions. The goal of sentiment analysis is to classify user opinions. The classification may be a simple polarity classification, i.e., negative or positive, or a more complex one, e.g., multiple ratings. Sentiment analysis can be used to process unstructured text written by a user to discover their interests, opinions, preferences, etc. to be included into their profile.

Trust and reputation management. Trust and reputation management is a set of algorithms and protocols for determining the trustworthiness of a previously unknown user in the context of his reliability in performing some action. For example, a reputation management system could be used for computing the trustworthiness of an online vendor who may or may not deliver the promised product once he receives payment. The reputation of a user is computed as an aggregate of the feedback provided by other users in the system. Trust and reputation information can be an important part of a user profile. It can convey the user’s trust in other users as well as his own reputation in various contexts. This information can be subsequently used as a basis for recommending trustworthy users and avoiding those who are untrustworthy. Trust and reputation management systems can function in conjunction with sentiment analysis for obtaining user opinions and then computing
trustworthiness and reputation.

**Machine learning.** Machine learning is a sub-field of artificial intelligence that aims to build algorithms that can make decisions not based on explicit programming but instead based on historical empirical data. An example often cited is the algorithmic classification of email into spam and non-spam messages without user intervention. In the context of personalization, machine learning can be used for learning user behavior by identifying patterns. Topics in machine learning include: supervised learning approaches, e.g., neural networks, parametric/non-parametric algorithms, support vector machines, etc.; and unsupervised learning approaches, e.g., cluster analysis, reduction of dimensionality, etc.

**Cluster analysis.** Cluster analysis is the process of classifying users (or any other objects) into smaller subgroups called clusters given a large single set of users. The clusters are formed based on the similarity of the users in that cluster in some aspect. Cluster analysis can be applied for discovering communities, learning membership of users in groups, etc. Cluster analysis can be considered as a sub-topic of machine learning.

### III. Privacy

On one hand, personalization with big data techniques is advantageous for providing better services as we have discussed above. On the other hand, big data poses a significant threat to user privacy. One can assume that an ethical and trustworthy service providing personalization would use the information collected about users with their explicit consent. However, services that are less inclined towards protecting user privacy, may use such data for a number of purposes which may not be approved by the user and which may result in loss of private information. An example is the selling of private information to third parties for a profit. The third parties may then use the private information of the users for commercial or even malicious purposes [2].

**A. What is privacy?**

Depending on the application and the targeted privacy requirement we can have different levels of information disclosure. Let’s take privacy preserving reputation systems (e.g. [5]) as an example. We can have five different levels for privacy depending on whether identities, votes and aggregated reputation score are disclosed and linked or not. For example, in the context of calculating the reputation of a user Alice by three other users Bob, Carol and David, which respectively have the votes +1, +1 and -1, the reputation system may disclose the following information to Alice.

- **Full disclosure.** All tuples (Bob,+1), (Carol,+1), (David,-1) as well as the aggregated score (+1 if sum is used) are known by Alice.
- **Permuted disclosure.** All voters Bob, Carol, David are known by Alice as well as the scores but permuted so Alice cannot determine who voted what.
- **Identity disclosure.** All voters Bob, Carol, David are known by Alice, however individual votes are hidden and only the aggregated score is known by Alice.
- **Votes disclosure.** All votes are known by Alice but the voters are hidden.

**Result disclosure.** No details are disclosed except the aggregated score.
**No disclosure.** An aggregated score for Alice is calculated but she does not have access to it.

More generally, we can subdivide privacy objectives in two:

**User anonymity**. The first objective is preserving user anonymity. In this setting, untrusted peers should not be able to link the identity of the user to the requests that they receive. For example, if Bob is navigating the web, any request that a content provider receives should not be linkable to the real user Bob. Information such as his IP address, Gmail identifiers, or any other such information which may help identify Bob should not be made available.

**Disclosure of private information about known users**. The second objective is preventing the disclosure of private information. Let’s take the same example of Bob searching on the web but desiring his age to be kept private. However, let’s suppose that he does not mind the origin of his query being revealed. In this case, privacy preservation does not necessarily require anonymity but rather providing guarantees that Bob’s age will not be disclosed.

**B. How can privacy be breached?**

Depending on the definition of privacy, different techniques can be used to breach privacy even within systems which intend to protect it. We identify two types of privacy attacks: (1) “protocol” attacks are those relying on protocol exchanges between peers, in particular using connection information (IP address, cookies, etc.), to identify users or information about them; (2) “statistical” attacks are those relying on statistical techniques (in particular statistical machine learning) to analyze flows of information reaching a peer and using automated reasoning techniques to deduce user identity or private characteristics.

1) **Protocol attacks:** Protocol attacks are those relying on the fact that since a user wants to obtain an information from a peer, then the peer will have to be contacted by some means. For example, a user wanting to access a web page on “looms” will have his browser making a request to the hosting server. Having been contacted the server has a trace of the user’s IP and knows that this IP has requested the page on looms. Protection from such attacks can be obtained by using proxies but this just moves the problem of trust from the content provider to the proxy provider. It is then the proxy which must be trusted. This very basic example gives an initial intuition on the fact that protecting from protocol attacks can get complex. Much research has been done on protecting anonymity from such protocol attacks.

2) **Statistical attacks:** Statistical attacks are those relying on the information which legitimately flows to a given peer. Even if users are protected by a privacy preserving protocol, the data which ends in the hands of a potentially malicious or curious peer may be used to break this anonymity. For example, to be able to find interesting documents for a user, a search engine must be provided with a search query. This query in itself provides information about the user from which it originates (be it only that he is interested in the topic of
the query). By correlating together the information that an untrusted peer has collected and linked together about a user, it can become possible to de-anonymize the user [6].

IV. THE EEXCESS USE-CASE

A. What is EEXCESS?

EEXCESS (Enhancing Europe’s eXchange in Cultural Educational and Scientific resources) (eexcess.eu) is a European Union FP7 research project that commenced in February 2013. The project consortium comprises of INSA Lyon (insa-lyon.fr), Joanneum Research (joanneum.at), University of Passau (uni-passau.de), Know-Center (know-center.tugraz.at), ZBW (zbw.eu), Bit media (bit.at), Archäologie und Museum Baselland (archaeologie.bl.ch), Collections Trust (collection-trust.org.uk), Mendeley (mendeley.com), and Wissenmedia (wissenmedia.de). In this section we present the EEXCESS project to illustrate how user profiling can benefit recommender systems particularly with the use of big data techniques. We also discuss the associated privacy issues and the approaches currently being considered in the project for tackling the privacy problem.

The main objective of EEXCESS is promoting the content of existing rich data sources available throughout Europe. While user context is more and more present, the current response of web search engines and recommendation engines to the massive amount of data found on the web has been to order query results based on some form of popularity. It is evident that the introduction of PageRank [7] in search engines has changed the landscape of online searching. However, this has lead to the effect of having large quantities of valuable content remaining simply unaccessed due to low levels of global popularity but at the same time being of high interest for a particular user. This unseen data is sometimes referred to as “long-tail content” in reference to the long-tail of a power-law distribution which in many cases characterizes the distribution of user interest in particular content.

It is this type of long-tail content that some of the EEXCESS partners are providing. This includes precise and rich content such as museum object descriptions, scientific articles, business articles, etc. Currently, this very specific content has trouble finding appropriate visibility, even though they would be invaluable in the appropriate contexts where fine-grained and precise information is sought for.

The aim of EEXCESS is to push such content made available by its partners to users when appropriate for them. However, this relies on having a precise understanding of a given user’s interests and their current context. Different levels of user profiling can help to characterize a user’s interests. In EEXCESS, precise user profiles will allow recommending the appropriate content found in multiple data sources.

B. Architecture

Figure 1 gives a sketch of the currently envisioned architecture for the EEXCESS project from a privacy perspective. From this perspective, EEXCESS is made of four components: (1) A plugin added to the user’s client whose role is to collect and transfer the user’s context, trigger recommendation requests and render them through rich visualizations, (2) a privacy proxy which collects the user’s privacy policy and ensures that it is respected, (3) a usage mining component allowing to identify common usage patterns and enrich user profiles accordingly, and (4) a federated recommender service composed of individual data-sources hosting a specific data collection. The circled numbers on the figure give the information flow when content is being recommended.

As suggested by the presence of a privacy-proxy, one major goal in EEXCESS is to respect its users’ privacy. In particular, no information about a user profile data should leak out of the system without the user’s consent. As will be discussed later, the project is faced with a conflicting situation in which disclosing more information will allow to improve recommendation quality but will also augment the risk if privacy leaks. The exact internals of the privacy proxy are among the works to be completed during the project’s time span. For simplicity, we consider the proxy-service as a single peer in this paper.

Let us consider a typical EEXCESS user scenario. Alice is an economist employed by a consulting firm. She is currently working on a business plan for one of her customers on a market which is new to her. As usual she uses her favorite search engine to investigate on the different actors of the market and in particular the potential competitors for her client. Fortunately, EEXCESS is connected to an economic database, and starts pushing to Alice relevant content from this database, which includes detailed descriptions of companies found in the target market of her client and strategic economic data. Alice requires that a high level of privacy is ensured by the system. In fact, she is legally-tied by a non-disclosure policy with her customer. In particular, it should not be learned that Alice’s customer is taking a move toward the new market.
C. Personalization

One of the major objectives of EEXCESS is recommending to its users, quality personalized content obtained from the EEXCESS partner data sources. To achieve this goal, fine-grained user-profiling will be an important part of the project and will consist of collecting sensitive data about the user. An important usage-mining component will be the collection or enrichment of user profiles using big data techniques as those described in Section II.

Of course, the user’s individual characteristics will be part of his profile. An EEXCESS user’s interests will either be interactively collected and/or completed using big data techniques implemented particularly by the usage mining service. User actions will be tracked by the EEXCESS plugin allowing to keep track of a user’s behavior. Among the partners of EEXCESS, Bit Media is an e-learning platform. In this case, it is clear that the user’s learning goals and current knowledge (e.g. in the form of courses already taken) will be part of the user’s profile. In EEXCESS, the user’s context will consist of information such as his current geo-location, the document or web page (both URL and content) he is working on, his browsing history, the navigation page which lead to the current page, etc.

To capture an even better understanding of the user, different big data techniques will be applied to further enrich his profile. For example, usage mining will try to identify usage trends, as well as information about the user’s unexpressed goals and knowledge. On-the-fly analysis of user interests, context, and expectations is also planned. Clustering techniques may be used to identify communities within EEXCESS users. This profiling and better understanding of the user has a unique goal in EEXCESS of providing the user a personalized experience of the system and in particular personalized recommendations. Indeed, the content of the EEXCESS partners being very specific (i.e. being in the long-tail of documents when ordered by popularity), having a fine-grained understanding of EEXCESS users is essential to link the correct users to the correct content.

In our example, the EEXCESS system will have collected significant information about Alice: her interests (economic information), some comprehension of her goal (writing a business plan), her knowledge (expert in economics), her context of work (information about her customer, the target market, the information she has already collected, etc.). Knowing as much as possible about Alice and her customer will allow the EEXCESS system to provide her with adapted recommendations. For example, instead of presenting general-purpose information about the market, the system will propose more detailed technical data which Alice needs and understands.

D. Privacy

Providing users with quality recommendations is a seemingly conflicting objective with the equally important goal of privacy preservation. Even a small amount of personal information may lead to identifying a user with high probability in the presence of side channel external data [3].

Returning to our example, it would be unacceptable to Alice that any information about herself or her customer leak out of the system. Alice’s project may even be so sensitive that even the fact that someone (without particularly knowing who) is setting up a business plan on the target market may be an unacceptable leak because it could lead to competitors taking strategic moves. This emphasizes the fact that preserving only anonymity may not be sufficient in some cases.

Therefore, for EEXCESS to be a success, many privacy-related challenges will have to be addressed.

Providing privacy guarantees. At all levels within the system user privacy guarantees must be given. This is most likely one of the hardest tasks. Indeed, as soon as information flows out of a system, sensitive information leaks become a risk. Solutions which may seem trivial, such as anonymization have been shown to be inefficient. A well known example showing that simple anonymization is insufficient to protect privacy is the de-anonymization of the data of the Netflix contest [3]. Furthermore, Dwork [8] has shown that the published results of a statistical database may lead to privacy breaches even for users who are not originally part of the database. These examples show the difficulties which will have to be overcome in order to provide a privacy-safe system. Furthermore, these works show that research on privacy has shifted from totally preventing privacy breaches to minimizing privacy risks. One of the difficulties to overcome in the EEXCESS project, is to ensure that the collection of information flowing out of the system to potentially malicious peers, limits the risks in breaching any of the users’ policies. It goes without saying that the attackers themselves very likely have access to big data techniques and that this aspect should be taken into account.

Flexible privacy policies. Users are different, in particular with respect to privacy. Some may not have any privacy concerns at all where as others may not want to disclose a single piece of information about themselves. For example, in one hypothesis, our user Alice may simply wish to remain anonymous. In another hypothesis, Alice may not be concerned by her identity being revealed, but wish that some information about her be kept private (e.g. she may wish to keep private that she is affected by a particular disease). One big challenge will be to define a policy model which allows for such flexibility and at the same time allows to ensure the policy is respected. Preventing direct disclosure of information marked private is quite straightforward. However, a real challenge is preventing the disclosure of the same information indirectly. Indeed, leaking other non-private information of a user’s profile can lead, through inference, to unwanted disclosures.

Evaluating trust and reputation. What user profile information is disclosed, or at which granularity it is disclosed, may depend on the trust (with respect to privacy concerns) that the user and/or the EEXCESS system has in the content provider. Calculating a content provider’s reputation and trustworthiness in a privacy preserving manner is thus an issue.

Let us consider the case of a user wishing to remain anonymous to all the recommenders. In this case, the attacker could be one of the content-providers trying to collect information about the user that it receives queries from. The EEXCESS privacy requirements for such a user would include:
**Content anonymity.** To guarantee privacy, the attacker should not be able to identify the user from the provided data. Therefore, the system should ensure that an attacker cannot deduce from the content of a request who it originated from.

**Request unlinkability.** If multiple queries can be linked together, even while having content-anonymity for each individual query, the combination of the two could reveal information about the user. Therefore, it should be required that the protocols guarantee that two independent requests originating from the same user are unlinkable.

**Origin unlinkability.** This should be feasible by anonymizing the origin of the request but under the condition that the origin is not revealed by the application level protocols. Therefore, we also need to guarantee that the application level protocols are privacy-preserving (i.e. an attacker cannot link a given request to the requesting user).

Respecting these three constraints is an ideal goal which requires limiting the information transmitted in each request. Such limitations have a high impact on the utility of the profile information disclosed. Thus the challenge is rather to find a balance between privacy and utility than to ensure complete privacy.

In information systems (such as recommender systems, statistical databases, anonymized datasets), the main goal of privacy preservation is to not reveal sensitive information about a single entity within the underlying data. This has been shown to be a difficult goal [8], [9]. In a survey on privacy in social networks, Zheleva and Getoor [10] describe some of the common approaches for preserving privacy: differential privacy and k-anonymity. In the context of recommender systems using collaborative filtering, an approach is to use big data techniques such as clustering to group users together in order to provide privacy [11], [12], [13] with the theory of k-anonymity.

In our particular setting, we are faced with a federated recommender system in which trusted and untrusted peers may exchange information. This requires that both the protocols for exchanging information and the content disclosed are privacy-safe. Furthermore, recommendations may not always be limited to a single recommendation technique among the peers. Each content source may wish to use its own approach. In the context of EEXCESS, few hypotheses can be made on the computational capacities or the background knowledge that an untrusted peer may have access to.

Our work in the EEXCESS project will include developing mechanisms for the definition of flexible user privacy policies, guarantees based on the user privacy policies for non-disclosure of private information, quantification of the risk of disclosing private information, mechanisms for exchange of information based on the reputation and trustworthiness of partners, as well as the definition of the relationship between the amount of information revealed and the quality of recommendations.

**E. EEXCESS specific challenges**

The EEXCESS context brings in different specific privacy related constraints and requirements. We discuss these constraints here and their impacts on privacy as similar “real world” constraints are likely not restricted to the EEXCESS case and should be considered when trying to reconcile personalization and privacy.

**Providers as blackbox recommenders** Currently, among the content providers, all provide an access to their content in the form of a standard search. Only one of them, namely Mendely, provides collaborative filtering. Therefore, in a first step, the focus has been put on recommendation through search. In the future, other forms of recommendation (collaborative filtering and hybrid recommendation) will be envisaged. However, in any case, the content-providers will be considered as black boxes in that it will not be known how recommendation is performed. This has an impact on privacy since the privacy-preserving mechanisms in place will have to be general enough to work with different recommendation solutions and cannot be limited to one form.

**Providers with existing profiles** Some of the content providers already have user-bases for which they may already have pre-calculated recommendations available. If privacy is limited to anonymization, then through EEXCESS, users will loose access to those recommendations as the recommenders of these providers will not be aware of the user they are sending recommendations for. Therefore, the privacy solutions studied should go beyond simple anonymity and allow users, at a minimum, to specify which providers they trust and are willing to share their information with.

**Provider recommenders needing feedback to quality** An important objective of recommender systems is to continuously improve themselves through user feedback. To this effect, it will be necessary for them to have access to such user feedback. However, this feedback should not be the source of privacy leaks. This is a challenge as many attempts towards anonymizing recommendation data have failed in that the data could be de-anonymized (Narayanan and Shmatikov 2008).

V. PERSONALIZATION, PRIVACY AND USERS

Personalization and privacy both aim to provide users a service that better fits their needs. Users therefore need to be implied in the process. In particular, users play an important role in information disclosure which, of course, has impacts on both personalization and privacy.

**A. Information disclosure**

Privacy is mostly all but a technical issue. Even though most of privacy-related work done within EEXCESS will be related to technical aspects in preserving privacy, it is also important to understand the user’s perspective. In a study on user behavior [14] have shown that user’s globally tend to disclose less information when users are faced with a system explicitly talking about privacy. The interpretation given is that when privacy issues are put in the focus, users tend to become more suspicious and therefore leak less information. This is quite a paradox as a system willing to be transparent finds itself disadvantaged with respect to one not mentioning privacy at all. However, the same work studies how to improve disclosure (compared to a system not mentioning privacy issues). Giving the same explanations to everyone will
lead to the tendency of users disclosing less because of the invocation of privacy. However, adapting explanations to the users can allow to improve disclosure. For example, within the test groups of [14], giving an explanation to men about what the data will be used for, and giving information to women about the percentage of users the data will be disclosed to, tended to globally improve disclosure.

1) Impacts of including users in the process: A system willing to successfully have its users disclose information willingly and at the same time respect their privacy must have solutions which adapt to them. Furthermore, giving high and precise control to users can on one hand show a will of transparency from the service provider, but on the other, this may make the system look too complex. Therefore, users should be provided with a system allowing them to set their privacy settings simply but without losing flexibility. To this effect, users should be able to specify their privacy concerns at a high level, but also be allowed more fine-grained settings.

Another important aspect to consider is providing users with elements to understand the effects of disclosing information. As discussed previously, this involves providing the appropriate explanations to the appropriate users. In the specific case of EEXCESS, the objective of user information disclosure is mainly to improve the quality of the recommendations for each user. This can, for example, be obtained through a tool allowing to compare results using different privacy settings.

Given a user’s preferences it is then necessary to have a system capable of optimizing the use of the disclosed information. In EEXCESS, this means that the quality of the recommendations should be maintained as close as possible to those that the user could have expected with a more detailed profile. Furthermore, providing recommendation quality will also rely on user profiling. Such deep user profiling entails many privacy concerns. Indeed, while users are likely to be interested in having very precise recommendations, they may not at the same time be willing that a third-party collects private information about them.

VI. RECONCILING PERSONALIZATION AND PRIVACY

A. Challenges

Currently, systems which provide personalization, function as a black box from the user’s perspective. Users do not know what is really collected about them, what about them is inferred by the system, with which other data sources their private data may be combined, what are their benefits of disclosure. Furthermore, faced with the multitude and growing number of external data sources, even limited disclosure of information to a given system may reveal enough about them for the same system to be able to infer knowledge they would have otherwise preferred remaining private. We list below some of the main categories of challenges that users face concerning their privacy in the existing big data systems.

Transparency. Users are often unable to monitor and follow precisely what information about them the system has collected. For example, it is common knowledge that different services, such as Google, Facebook, Amazon, etc., use big data analytics to provide personalization in many of their services. However, it is not always transparent to users what information has been collected, inferred and how it is used by whom. Even if these services wish to provide more transparency it is often technically challenging to provide tools to visualize complex processing and manipulation (and in particular aggregation) of user information.

Control. Users are often unable to express their private information disclosure preferences. This can either be due to the unavailability of such options, the complexity of the provided tools or even their unawareness of privacy issues. They should be able to specify what is disclosed and how detailed the disclosure should be as well as to whom it is disclosed. A big challenge for control is that the more fine-grained privacy settings are the more complex and time consuming it becomes for users to set them. Furthermore, not all users have the same level of requirements, some are willing for such fine-grained control, whereas others would be satisfied with simpler high level control.

Feedback. Users often have difficulties understanding the impacts of disclosing or not disclosing certain pieces of information on personalization. Personalization is impacted by the type, quantity and quality of information users provide. It is difficult for users to clearly perceive how their inputs impact personalization. This is amplified by the fact that often, these impacts are differed in time and their effects come only later. Also, in many cases, when they do perceive the advantages or lack of value of providing some piece of information, it is long after they have provided it. To make things worse, once the information is released, it is hard for it to be completely retracted.

Re-identification. Because of big data techniques, such as machine learning, very few discriminant data allow to (re)identify the user at the origin of a request. For example, it is possible for a search engine to (re)identify some queries sent by a single user among all the queries. This is true even if the user is connected to the search engine via an anonymous network such as TOR [15]. This is done by using the content of the messages (rather than who they are coming from) and using classification techniques to (re)identify their likely origin. This suggests that anonymous networks or query shuffling to guarantee unlinkability between users and their requests may not be enough. Therefore, within the context of personalization we are faced with a paradox: on one hand we want to adapt results to specific users, which requires discriminating the user from the others, and on the other hand, to preserve user privacy we should rather not discriminate them.

Discovery. Big data techniques can be utilized for discovering previously unknown information about a given individual. For example, through statistical reasoning, having access to the list of visited web sites may reveal the gender of the users even if they have not given them explicitly.

Privacy and utility balance. On one hand, personalization pushes towards providing discriminant data (the more the better) about users whereas privacy pushes to have non-discriminant data (the less the better). However, many personalization techniques rely on using data from similar users. If groups of similar users are sufficiently wide, it becomes difficult to distinguish users among these groups.
Collusion. Collusion between peers is another risk for privacy. Indeed, the information which may not be individually discoverable through two uncombined sources of information, when combined through collusion, could lead to new discoveries and therefore privacy breaches.

B. Impacts within the EEXCESS use-case

Privacy has multiple impacts on the conception of systems heavily relying on personalization. Much depends on the trustworthiness of the peers, but most of all, the legal entities running these peers. In the case of EEXCESS, the architecture of the system and used recommendation algorithm is highly dependent on the trust put in the legal entity which will host the EEXCESS software. If the federated recommender component could be hosted by possibly untrustworthy peers, then it could be required that the component required being distributed and/or make use of cryptographic solutions.

We can summarize the different impacts privacy has on personalization within the EEXCESS system as follows:

1) Adapting personalization algorithms: Providing privacy-preserving personalization implies adapting existing personalization algorithms. Many approaches include cryptographic mechanisms, distribution over a network of peers, working with partial and incomplete data, working with groups of users or pseudonyms, etc.

2) Architectural impacts: Privacy-preservation can often not be limited to inventing a new version of an algorithm. It has impacts on the global architecture of the privacy preserving system. Indeed, many privacy-preservation mechanisms rely on the fact that all the data does not reside on a single peer. This is particularly true to allow relaxing trustworthiness assumptions on some or part of the peers. Figure 2 gives different options of the trustworthiness assumptions within the EEXCESS architecture. Depending on the chosen trust scenarios, the privacy preserving strategies will require being adapted. For example, if privacy proxy cannot be trusted (scenario (e) of figure 2), this will impose that the proxy be a distributed component among multiple non-colluding authorities to ensure that a single authority does not have all the information.

3) Making privacy-preservation dynamic: Finally, taking user preference into the privacy preservation mechanisms requires that the personalization algorithms dynamically adapt to each user. In particular, the information provided for two similar users but with different privacy preferences will imply that the data available for each of those users be not as detailed. For example, users may provide some information at different levels of granularity. One user may allow providing a complete birth date whereas another may only allow revealing her age range.

C. Existing solutions

There is a significant amount of research currently in progress to achieve the goal of preserving user privacy while collecting personal information. Big data techniques offer excellent opportunities for more accurate personalization. However, privacy is an issue that can hinder acceptance by users of personalization with big data techniques. Therefore, there is a need to develop big data techniques that can collect information for user profiles while respecting the privacy of the users. Privacy preserving big data techniques for personalization would raise the confidence of users toward allowing services to collect data for personalization purposes. Below, we list some of the works on privacy preservation in domains related to big data.

1) Privacy preserving reputation management: Much work has been done in field of privacy preserving reputation management. A privacy preserving reputation management system operates such that the opinions used to compute a reputation score remain private and only the reputation score is made public. This approach allows users to give frank opinions about other users without the fear of rendering their opinions public or the fear of retaliation from the target user. Privacy preserving reputation management systems for centralized environments include those by Kerschbaum [16] and by Bethencourt et al. [17]. The system by Kerschbaum introduces the requirement of authorizability, which implies that only the users who have had a transaction with a ratee are allowed to rate him even though rating is done anonymously. Bethencourt’s system lets a user verify that the reputation of a target user is composed of feedback provided by distinct feedback providers (implying no collusion) even when users are anonymous. Hasan et al. [5], [18] propose privacy preserving reputation management systems for environments where the existence of centralized entities and trusted third parties cannot be assumed. Current privacy preserving reputation management systems still face a number of open issues. These include attacks such as self-promotion and slandering, in which a user either submits unjustified good opinions about himself or unwarranted bad opinions about a competitor.

2) Differential Privacy: In the domain of statistical databases, a major shift occurred with the work of Dwork and the introduction of differential privacy [8], [19], [20]. Through a theoretical framework, the authors demonstrate that, as soon as we consider external knowledge, privacy breaches can occur even for people who do not participate in a statistical database. This has introduced a shift in the way to perceive privacy. The objective is no longer to preserve privacy in an absolute manner, but rather limit the risk of increasing the privacy breach for a participant of a statistical database. To this effect, differentially private mechanisms are those that ensure that the statistical outputs of two databases which are only different by a single participant return similar statistical results. This most often consists in adding sufficient noise to the outputs. Even though there are situations in which differential privacy is attainable, in particular count queries, there are many constraints imposed by differential privacy [21], [22]. In particular, in situations which should allow multiple queries, noise must be augmented proportionally to the number of queries to prevent noise reduction techniques to be applied. However, adding too much noise can deprive the outputs of the system of any utility. Therefore much research is ongoing to evaluate the trade-offs between privacy and utility [23]. However, in practice, differential privacy can render some subsets of the randomized data less useful while poorly preserving the privacy of specific individuals. This has been demonstrated for instance in [22]. Thus, privacy preserving techniques still have much to achieve in order to render personal information of users truly private.
3) K-anonymity: Recommenders need to massively gather past user interactions and their ratings about objects that they were concerned with. This allows them to propose a selection of predicted objects to a current user, based on profile similarity analysis with the current user, using techniques like collaborative filtering. While this allows having good recommendation quality, it also creates user privacy concerns. K-anonymity is one of the well-known techniques to preserve user privacy. The recommender in this case should ensure that each selected object has been selected by at least k users and that each object has been rated similarly by at least K users. This allows avoiding structured-based and label-based attacks respectively [24]. Several methods have been proposed to ensure k-anonymity among them, we can cite [25], [24], [26], [27], [28]. Many solutions are aimed at resolving k-anonymity problem in databases [25], [27], [28], [24], [26] both proposed using k-anonymity for privacy preserving recommenders. In both, past user ratings are represented using a bi-partite graph, where nodes are subdivided into user nodes and object nodes. A graph edge represents the rated selection of an object by a user. Projecting the graph on a single user gives the knowledge that the system has about that user rate and selections. The k-anonymity is obtained then by padding the graph cleverly so that a user clustering with less recommendation accuracy could be obtained. Whereas most solutions proposed for recommenders are based on a centralized gathering of user rates, [26] propose a user-centric distributed and anonymous solution to gather useful information to make recommendations. Interestingly, recent work has shown that it can be linked with differential privacy under certain circumstances [29].

4) Anonymization protocols: [30] introduced a routing protocol allowing the anonymization of communications between two peers by shuffling messages and there for disabling a server from knowing where a given message came from. The onion router [15] improves anonymity by using cryptography. A client message is encrypted multiple times with the with the keys of the peers of the routing path. This protocol preserves the target server from knowing the address of the client as long the intermediate peers do not collude. However, it is often possible to still identify the original user through information provided within the message itself. This is typically the case of web cookies and/or protocol headers. Solutions exist through browser extensions such as FoxTor or TorButton cookies and headers. However, the body of the message itself (e.g. a search query) which is required for the target server to provide a response (e.g. search results) itself reveals information about the user which in some cases may lead to user re-identification [6].

5) Data obfuscation: To tackle attacks based on the content of the message, works in the literature have proposed to rely on data obfuscation. Different works have suggested such an approach in the case of web search [31], [32]. In the case of search queries, seen as a collection of terms, the idea is to build an obfuscated query by adding extra decoy terms to the query. The obfuscated query is sent to the search engine which can therefore not know what the original query was. Search results are then filtered by the client in order to restore the accuracy of the original request.

6) User groups: Based on the ideas of [11], their exist many works relying on providing personalization for groups of similar users rather than the users themselves. For example, [33], [34], propose aggregating data of multiple users belonging to similar interest groups. The target group a user belongs to is calculated locally. A group profile is built anonymously using distributed and cryptographic techniques.

D. Discussion

Ideally the user would like to obtain quality and personalized recommendations without revealing anything about himself. Attaining such an objective means ensuring that a user remains anonymous with respect to the peers he considers non-trustworthy. Works have shown that in some restricted cases anonymization is possible [11], [8], [19]. This however often comes at the cost of quality or utility of the disclosed information [21], [29], [3], [22]. It may also be the case that users do not necessarily require anonymity (for example, in social networks), but rather have control over what is disclosed or not disclosed.

In the light of the existing work, the current privacy challenges that the EEXCESS will face can be summarized as described hereafter.

1) Adapting privacy to user preferences: Users have different views on what should be private and give privacy varying importance. Some may prefer having very good personalization whereas others favor privacy. Privacy preservation is already a challenge in itself. Taking user privacy preferences requires that privacy algorithms be able to dynamically adapt to the user’s preferences.
2) Limiting the impacts of privacy on personalization quality: Ideally, privacy-preservation mechanisms should not impact the quality of personalization obtained from the user. However, this is likely not easily achievable. A less restrictive requirement is that the privacy-preservation mechanisms should minimize the impacts of privacy-preservation on the quality of personalization. This implies, of course, being capable of measuring such quality which in itself could be a challenge.

3) Providing privacy-preserving user feedback: In EEXCESS, one form of personalization envisaged is through recommendations. These techniques require user feedback on what they have previously found useful to them. This feedback could imply privacy leaks if not dealt with appropriately. Therefore, such profiling mechanisms should also be required to be privacy-preserving.

4) Preserving privacy in the context of user profiling: Finally, EEXCESS will rely on user profiling to provide personalization. Such profiling will accumulate much data which could allow identifying users and private information about them. Therefore, these profiling mechanisms should also be adapted to preserve privacy.

VII. CONCLUSION

In this position paper, we discussed the challenges raised when building systems which require at the same time a deep level of personalization and a high level of user privacy. Personalized systems can provide high value to users. This is particularly true when providing personalized recommendations of long-tail content. Big data analysis techniques play an important role in making such personalization possible. On the other hand, this raises the issue of respecting a given user’s privacy. Big data may even increase this risk by providing attackers the means of circumventing privacy-protective actions. We illustrated these issues by introducing the challenges raised by EEXCESS, a concrete project aiming both to provide high quality recommendations and to respect user privacy.

ACKNOWLEDGMENT

The presented work was developed within the EEXCESS project funded by the EU Seventh Framework Program, grant agreement number 600601.

REFERENCES


A FORECASTING APPROACH FOR DATA ALLOCATION IN SCALABLE DATABASE SYSTEMS

Shun-Pun Li, Man-Hon Wong
Department of Computer Science and Engineering
The Chinese University of Hong Kong
{spli, mhwong}@cse.cuhk.edu.hk

Abstract

In cloud computing environment, database systems have to serve a large number of tenants instantaneously and handle applications with different load characteristics. To provide a high quality of services, scalable distribute database systems with self-provisioning are required. The number of working nodes is adjusted dynamically based on user demand. Data fragments are reallocated frequently for node number adjustment and load balancing. The problem of data allocation is different from that in traditional distributed database systems, and therefore existing algorithms may not be applicable. In this paper, we first formally define the problem of data allocation in scalable distributed database systems. Then, we propose a data allocation algorithm, which makes use of time series models to perform short-term load forecasting. For online applications, probably, there are observable access patterns and peak load hours. With an accurate load forecasting, node number adjustment and fragment reallocation can be performed in advance to avoid node overloading and performance degradation due to fragment migrations. In addition, excessive working nodes can be minimized for resource-saving. For verifying the feasibility of our forecasting approach, time series analysis is conducted on real access logs. Simulations are performed to evaluate and compare the proposed algorithm with the traditional threshold-based algorithm.

Keywords: data allocation, scalable database systems, load forecasting, load balancing, resource-saving

1. INTRODUCTION

In traditional distributed database systems, the problem of data allocation has been well-defined (Apers, 1988). Designing a distributed database is said to be an optimization problem on data fragmentation (also known as data partitioning) and data allocation. To ensure all nodes in a distributed system work independently and efficiently, data fragmentation is performed such that fragments can be stored in different nodes. Then, allocation of the fragments is considered as to distribute the workloads and reduce the data transfer cost. Since there is no query router and it is assumed that fragments can be accessed through any node, an extra transfer cost is required if the fragment is not located in the current node. The problem of data allocation is then focused on storing the fragments at the nodes in which the fragments are most frequently accessed. Reallocation of the fragments is performed only when there is a change in access patterns. However, the problem becomes complicated in scalable distributed database systems (Aguilera et al., 2007).

In literature, two scalable distributed database systems were proposed for cloud platforms (Das et al., 2010; Curino et al., 2011). From their designs, we found that assumptions for traditional distributed database systems are no longer valid. Instead of having a fixed number of nodes, the number of working nodes in scalable distributed database systems is adjusted dynamically based on user demand. Fragments are reallocated frequently for node number adjustment and load balancing. A master node is present and it plays an important role for managing and monitoring the whole system. Due to these changes, data allocation algorithms for traditional distributed database systems may not be applicable. Our goal is changed to minimize the performance degradation
resulted from node number adjustment and fragment reallocation while achieving load balancing and resource-saving.

In this paper, we first formally define the problem of data allocation in scalable distributed database systems. Then, we propose an algorithm for the problem. The algorithm makes use of time series models to perform short-term load forecasting (STLF) and dynamically reallocate data fragments to balance the workload of the system. STLF is a technique adopted in practice and its importance has been reviewed by Gross (1987). The time series model used in this paper is the autoregressive integrated moving average (ARIMA) model, which is one of the popular models in STLF. The advantages of using time series models are: (i) future workloads can be easily estimated; (ii) overloading of any nodes can be predicted and avoided; (iii) fragment reallocation and node number adjustment can be performed in advance; (iv) performance degradation due to fragment migrations can be minimized. For online applications, probably, there are observable access patterns and peak load hours. By applying automatic forecasting techniques (Hyndman & Khan-dakar, 2008), future workloads can be estimated in runtime without human tuning. Specifically, this paper makes the following contributions:

- We give a formal definition of data allocation problem in scalable distributed database systems.
- We propose an efficient data allocation algorithm, which gives a good performance with accurate load forecastings.
- We show that the proposed algorithm is a generalization of threshold-based algorithms. It can be reduced to a general threshold-based algorithm.

The rest of this paper is organized as follows. Section 2 gives a general model of scalable distributed database systems. Section 3 formulates the problem. Section 4 presents the methodologies used. Section 5 describes the proposed algorithm and proves the correctness. Section 6 reports our experimental results. Section 7 presents related works and Section 8 contains our conclusion.

2. Model

To generalize the problem, we regard a scalable distributed database system as an abstract model with three layers, as shown in Figure 1. The bottom layer consists of a set of data fragments, which store the data of different applications. Fragments are assumed to be non-replicated. The middle layer consists of a set of database nodes that are independent database servers for storing data fragments and processing user queries. Lastly, the upper layer is a master node for routing user requests to the corresponding database nodes. When there is a request sent from an application for retrieving data stored in a particular fragment, the master node first finds the database node that owns the required fragment. Then, it provides the location of the node to the application such that the application can communicate with the node directly for further query processing. In addition, the master node monitors the health and proper working of database nodes within the system. New nodes will be added if there are too many requests. Excessive nodes will be removed if the workload to systems is decreasing.

3. Problem Formulation

Based on the model described in the previous section, we formalize the problem of data allocation in this section. Let the set of database nodes and the set of data fragments be $\Omega_N = \{N_1, N_2, \ldots, N_n\}$ and $\Omega_F = \{F_1, F_2, \ldots, F_m\}$ respectively, where $n$ denotes the size of $\Omega_N$ and $m$ denotes the size of $\Omega_F$. $n$ and $m$ are not fixed values and may vary with time. The relationship between $\Omega_N$ and $\Omega_F$ is defined by the fragment allocation matrix $A$, as described below.

**Definition 1. (Fragment Allocation Matrix) $A$ is an $n \times m$ matrix, where $n = |\Omega_N|$ and $m = |\Omega_F|$. An element $a_{ij} = 1$ if fragment $F_j$ is owned by node $N_i$; otherwise, $a_{ij} = 0$.**
Example 1. The matrix shown in Figure 2(a) represents the allocation of a set of fragments \( \{F_1, F_2, F_3, F_4, F_5\} \) over a set of nodes \( \{N_1, N_2, N_3\} \). We say fragments \( F_1 \) and \( F_3 \) are owned by node \( N_1 \) as \( a_{11} = 1 \) and \( a_{13} = 1 \). Fragments \( F_2 \) and \( F_5 \) are owned by node \( N_2 \) as \( a_{32} = 1 \) and \( a_{35} = 1 \). Fragment \( F_4 \) is owned by \( N_2 \) as \( a_{24} = 1 \).

The workload of a fragment is represented by the number of requests for the fragment in a unit of time where a unit of time can be a second, a minute, an hour, or a week. Formally, we define the fragment workload as follows.

Definition 2. (Fragment Workload) The workload of fragment \( F_j \) at time \( t \), denoted by \( W_{F_j,t} \), is the number of requests for \( F_j \) between time points \( t - 1 \) to \( t \).

Since each node may contain more than one fragment, the workload of node \( N_i \) at time \( t \), denoted by \( W_{N_i,t} \), is the sum of workloads of its fragments at time \( t \).

\[
W_{N_i,t} = \sum_{j=1}^{m} W_{F_j,t} \times a_{ij}
\]

Intuitively, the workload to the database system is the sum of workloads of its nodes.

Example 2. Table 1 shows the workloads of \( F_1, F_2, ..., F_5 \) from time \( t = 1 \) to \( t = 5 \). From Figure 2(a), \( F_1 \) and \( F_3 \) are owned by \( N_1 \). Therefore, \( W_{N_1,1} = W_{F_1,1} + W_{F_3,1} = 5 \).

In reality, a node can only handle a limited number of requests within a unit of time. It is referred to be the maximum throughput of the node. We say a node is overloaded at time \( t \) if the workload at time \( t \) is greater than its maximum throughput. To avoid node overulings, fragment migrations are performed. However, frequent migrations may result in a high cost. The migration cost is defined as follows.

Definition 3. (Migration Cost) The cost for migrating a fragment from one node to another node is the number of requests suspended during the migration.

Example 3. Suppose the maximum throughput of a node is 10. At time \( t = 1 \), \( N_3 \) is overloaded, i.e. \( W_{N_3,1} = 11 \) (See Figure 2(a) and Table 1). To reduce the workload of \( N_3 \), either \( F_2 \) or \( F_3 \) has to be migrated to another node. We say the migration cost of \( F_2 \) is less than that of \( F_5 \) since \( W_{F_2,1} < W_{F_5,1} \). Therefore, \( F_2 \) is migrated to \( N_2 \), and the fragment allocation matrix is updated accordingly, as shown in Figure 2(b).

Besides node overulings, migration costs may result from increasing or decreasing the number of working nodes. We generalize the process as node number adjustment.

Example 4. Suppose the maximum throughput of a node is 10. At time \( t = 2 \), the workload to the system is 20 (See Table 1). Two nodes are able to handle all requests. Therefore, \( F_2 \) and \( F_5 \) are migrated to \( N_1 \) and \( N_2 \) respectively, as shown in Figure 2(c). \( N_3 \) is removed afterward.

In our design, node number adjustment and fragment reallocation are performed at time \( t \) for load balancing and resource-saving at time \( t + 1 \). The problem we are going to solve is then defined as follows.

Problem Definition. Given a set of nodes \( \Omega_N \) and a set of fragments \( \Omega_F \) distributed over \( \Omega_N \), the problem is to reallocate the fragments and adjust the number of working nodes at any time \( t \) with minimum migration costs such that there is no overloaded or excessive working node at time \( t + 1 \).
4. Methodology

Before going into the details of the proposed algorithm, methodologies for the problem are described in this section. For better understanding of our approach, basic knowledge on time series load forecasting is presented.

4.1 General Approach

In reality, we do not know the things that have not yet happened. The simplest solution adopted by traditional approaches is to use threshold testing. When the workload of a node is greater than a threshold value, some fragments belonged to the node are migrated to other nodes to reduce the workload of the node. Namely, fragment migrations are performed only when a node is already overloaded. The migration cost can be very high and an unnecessary migration cost may be generated, as shown in the following scenarios.

Scenario 1. Suppose the maximum throughput of a node is 10. At time \( t = 1 \), \( N_3 \) is overloaded, i.e. \( W_{N_3} = 11 \) (See Figure 2(a) and Table 1).\( F_2 \) has to be migrated to either \( N_1 \) or \( N_2 \). If \( F_2 \) is migrated to \( N_1 \), \( N_1 \) will become overloaded at time \( t = 2 \), and therefore further migrations are required. However, no node will become overloaded at time \( t = 2 \) if \( F_2 \) is migrated to \( N_2 \).

Scenario 2. Suppose the maximum throughput of a node is 10. At time \( t = 2 \), the workload to the system is 20 (See Table 1). Since two nodes are able to handle all requests, \( N_3 \) is removed (See Figure 2(c)). However, at time \( t = 3 \), the workload to the system increases to 23. A node has to be added back and fragments need to be reallocoted.

For the prototype system developed by Das et al. (2010), an observation period is required in a threshold-based algorithm for checking whether a node keeps overloading. Since a node will only be added or removed if the change in workloads is observed over a period of time, the algorithm results in a longer overloading time and may not be proper if the trend changes just after the observation period. In our design, we reduce the chance of generating unnecessary migration costs by applying forecasting techniques.

4.2 STLF and ARIMA model

Short term load forecasting (STLF) refers to the prediction of the system load over an interval ranging from one hour to one week. For classical approaches, STLF is mainly based on time series models. In this paper, the auto regression integrated moving average (ARIMA) model is used. It takes three parameters, \( p, d \) and \( q \), which refer to the orders of AR, integrated and MA parts of the series. By definition, a series \( \{Z_t \} \) with an ARIMA\((p, d, q)\) model is expressed as:

\[
\phi(B)(1 - B)^d Z_t = \psi(B) a_t
\]

\( B \) is a backshift operator, which operates on a term to produce the previous term.

\[
B^k Z_t = Z_{t-k}
\]

\( a_t, a_{t-1}, \ldots \) are white noise error terms, which are independent and identically distributed (i.i.d.). The AR operator \( \phi(B) \) and the MA operator \( \psi(B) \) share no common terms and are expressed as:

\[
\phi(B) = 1 - \phi_1 B - \ldots - \phi_p B^p \\
\psi(B) = 1 - \psi_1 B - \ldots - \psi_q B^q
\]

The parameter \( p \) refers to the dependence on the past terms. An ARIMA\((p, 0,0)\) is a pure \( \text{AR}(p) \) model in which the current term is a linear combination of the previous terms.

\[
Z_t = \phi_1 Z_{t-1} + \phi_2 Z_{t-2} + \ldots + \phi_p Z_{t-p} + a_t
\]

The parameter \( q \) refers to the dependence on the white noise error terms. An ARIMA\((0,0,q)\) is a pure \( \text{MA}(q) \) model in which the current term is a linear combination of white noise error terms.

\[
Z_t = a_t - \psi_1 a_{t-1} - \psi_2 a_{t-2} - \ldots - \psi_q a_{t-q}
\]

The parameter \( d \) refers to the degree of differencing. By differencing, a non-stationary series can be transformed into a stationary series.

For online applications, workloads are usually influenced by seasonal effects. To model fragment workloads precisely, a seasonal ARIMA (SARIMA) model should be used. An ARIMA \((p, d, q) \times (P, D, Q)\) model is a SARIMA model with non-seasonal orders \( p, d, q \), seasonal order \( P, D, Q \) and seasonal period \( s \). It is expressed as:

\[
\phi(B) \Phi(B) (1 - B)^d (1 - B^s)^D Z_t = \psi(B) \Psi(B) a_t
\]
where $\Phi(B)$ and $\Psi(B)$ are seasonal AR and MA operators, which take $s$ as the time lag and are expressed as:

$$\Phi(B) = 1 - \Phi_1 B^s - \cdots - \Phi_p B^{sp}$$
$$\Psi(B) = 1 - \Psi_1 B^s - \cdots - \Psi_q B^{sq}$$

Interested readers may refer to the book by Brockwell and Davis (2002) for a more detailed description of the model.

### 4.3 Design Rationale

We assume that workloads can be modeled as observed time series. For each fragment, we model its workloads as an ARIMA model and estimate the future workloads by the $k$-step ahead forecasting method.

**Definition 4. ($k$-step Ahead Forecasting Workload)** The $k$-step ahead forecasting workload of $F_j$, denoted by $\hat{W}_{F_j,t}(k)$, is the estimated value of $W_{F_j,t+k}$ at time $t$.

**Example 5.** Suppose the series $\{W_{F_{1,t}}\}$ in Table 1 follows an ARIMA(2,0,0) model with $\phi_1 = 0.4$ and $\phi_2 = 2$:

$$W_{F_{1,t}} = 0.4W_{F_{1,t-1}} + 2W_{F_{1,t-2}} + a_t$$

At time $t = 2$, we estimate the values of $W_{F_{1,3}}$ and $W_{F_{1,4}}$ as follows:

$$\hat{W}_{F_{1,2}}(1) = 0.4W_{F_{1,2}} + 2W_{F_{1,1}} = 4.8$$
$$\hat{W}_{F_{1,2}}(2) = 0.4\hat{W}_{F_{1,2}}(1) + 2W_{F_{1,2}} = 5.92$$

From the above example, we can see that future workloads are obtained from a linear equation. Once we have the model, the computation is straightforward and can be completed in linear time.

**Definition 5. ($k$-step Ahead Forecasting Error)** The $k$-step ahead forecasting error of fragment $F_j$, denoted by $e_{F_j,t}(k)$, is the difference between the actual value $W_{F_j,t+k}$ and the estimated value $\hat{W}_{F_j,t}(k)$.

**Example 6.** Consider the estimated values shown in the previous example and the exact values shown in Table 1. The forecasting errors of $\hat{W}_{F_{1,2}}(1)$ and $\hat{W}_{F_{1,2}}(2)$ are calculated as follows:

$$e_{F_{1,2}}(1) = |5 - 4.8| = 0.2$$
$$e_{F_{1,2}}(2) = |5 - 5.92| = 0.92$$

For $k$-step ahead forecasting, the accuracy decreases when $k$ goes up. Hence, $k$ should not be too large. We denote the maximum value of $k$ as a user-defined constant $\omega$. Based on the estimated workloads, our first goal is to reduce the unnecessary migration costs generated within the time interval $t+1$ to $t+\omega$. If an action is intended to generate an unnecessary migration cost within the time interval, the action will be forbidden and not be performed. Secondly, we want to stabilize the nodes during fragment reallocation.

**Definition 6. (Stable Node)** A node is stable if the workload of the node keeps constant with time.

To measure the stableness, we transform the estimated workloads into load trends, as defined below.

**Definition 7. (Load Trend)** The load trend of fragment $F_j$ at time $t$, denoted by $\hat{G}_{F_j,t}$, is the gradient of the regression line of $\hat{W}_{F_j,t}(k)$ for $1 \leq k \leq \omega$.

In mathematics, given a set of points, the gradient can be obtained by the ordinary least squares method. Therefore, we express the estimated load trend of fragment $F_j$ as follows:

$$\hat{G}_{F_j,t} = \sum_{k=1}^{\omega} \left[ \left( \frac{1}{\omega} \sum_{i=1}^{\omega} v \right) \left( \hat{W}_{F_j,t}(1) - \frac{1}{\omega} \sum_{i=1}^{\omega} \hat{W}_{F_j,t}(v) \right) \right]$$

Based on the estimated load trend of fragments, we can further obtain the estimated load trend of a node, as proved in Lemma 1.

**Lemma 1.** The estimated load trend of node $N_i$ is the sum of the estimated load trends of all fragments owned by $N_i$.

**Proof.** Suppose node $N_i$ owns a set of fragments $F_1, F_2, \ldots, F_u$.

$$\hat{G}_{N_i,t} = \sum_{k=1}^{\omega} \left[ \left( \frac{1}{\omega} \sum_{i=1}^{\omega} v \right) \left( \sum_{j=1}^{u} \hat{W}_{F_j,t}(1) - \frac{1}{\omega} \sum_{i=1}^{\omega} \hat{W}_{F_j,t}(v) \right) \right]$$

$$= \sum_{i=1}^{\omega} \left[ \left( \frac{1}{\omega} \sum_{i=1}^{\omega} v \right) \left( \sum_{j=1}^{u} \hat{W}_{F_j,t}(1) - \frac{1}{\omega} \sum_{i=1}^{\omega} \sum_{j=1}^{u} \hat{W}_{F_j,t}(v) \right) \right]$$
In other words, a node is stable if its load trend is close to zero. Figure 2(a) shows that nodes with a non-positive load trend are closer to zero. Therefore, we say the load trend of an overloaded node is negative, i.e. the workload of the node is decreasing. Probably, removing fragments in a reasonable way. Suppose the nodes with a non-positive load trend. Suppose a node is represented by an M/M/1 queue, which assumes that the request rate (workload per unit of time) follows a Poisson distribution with mean $\lambda$ and the service time (unit of time over maximum throughput) follows an exponential distribution with mean $1/\mu$. A node is said to be stable if $\rho = \lambda/\mu < 1$, i.e. the workload of the node is less than the maximum throughput. Given the measured service rate $\hat{\mu}$ (maximum throughput) and the desired $\rho$, we compute the confidence interval $\rho_u$ and $\rho_l$ by $F$-test (Lilliefors, 1966). Then, $\theta_u$ and $\theta_l$ are obtained as follows:

$$\theta_u = \lambda_u = \rho_u \hat{\mu}$$
$$\theta_l = \lambda_l = \rho_l \hat{\mu}$$

Positive load trend is preferred, and we select destination node $N_i$ for fragment $F_j$ based on the following constraint:

$$\hat{G}_{N_i,t} + \hat{G}_{F_j,t} \leq 0$$ (2)

Recall that excessive working nodes have to be removed for saving resources. We further define an upper bound $\theta_u$ and a lower bound $\theta_l$ for the workload of a node. For flexible fragment reallocation, it is assumed that the gap between $\theta_u$ and $\theta_l$ is large enough such that $\theta_u - \theta_l > W_{F_j,t}$ for any fragment $F_j$ at any time $t$. We say there is no excessive working node if all nodes are working at a rate within $\theta_u$ and $\theta_l$. The minimum number of working nodes required for time point $t$, denoted by $n_t$, is calculated as follows:

$$n_t = \left\lceil \sum_{j=1}^{m} W_{F_j,t} / \theta_t \right\rceil$$

The formula determines how many nodes are required such that all nodes can work at a rate near to the lower bound $\theta_l$. We use $\theta_t$ as the denominator because we are less willing to see a node being overloaded than underloaded.

### Example 7

Table 2 summarizes the workloads of $N_1$, $N_2$, and $N_3$ based on Table 1 and the fragment allocation matrices shown in Figure 2(a) and Figure 2(d). For the allocation based on Figure 2(d), the load trends are closer to zero. Therefore, we say the nodes are more stable.

**Table 2. Workloads under different allocation schemas**

<table>
<thead>
<tr>
<th>$t$</th>
<th>$W_{N_1,t}$</th>
<th>$W_{N_2,t}$</th>
<th>$W_{N_3,t}$</th>
<th>$W_{N_1,t}$</th>
<th>$W_{N_2,t}$</th>
<th>$W_{N_3,t}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>5</td>
<td>11</td>
<td>7</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>4</td>
<td>9</td>
<td>6</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>2</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>4</td>
<td>10</td>
<td>8</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>7</td>
</tr>
</tbody>
</table>

$\hat{G}_{N_1,t} = 1.0\, \hat{a}_{11}$

$\hat{G}_{N_2,t} = 0.2\, \hat{a}_{21}$

$\hat{G}_{N_3,t} = -0.5\, \hat{a}_{31}$

In short, we express the estimated load trend of node $N_i$ as follows:

$$\hat{G}_{N_i,t} = \sum_{j=1}^{m} \theta_i \hat{a}_{ij}$$

We say $N_i$ is more stable than $N_i$ if $|\hat{G}_{N_i,t}| < |\hat{G}_{N_i,t}|$.

In other words, a node is stable if its load trend is close to zero.

During fragment reallocation, fragments are removed from overloaded nodes. Besides reducing the workload, stabilization can be achieved by removing fragments in a reasonable way. Suppose the load trend of an overloaded node is negative, i.e. the workload of the node is decreasing. Probably, removing fragments with negative trend may make the load trend of the node closer to 0. Therefore, we select fragment $F_j$ from overloaded node $N_i$ based on the following constraint:

$$|\hat{G}_{N_i,t} - \hat{G}_{F_j,t}| \leq |\hat{G}_{N_i,t}|$$ (1)

On the other hand, we can avoid generating unnecessary migration costs by migrating fragments to the nodes with a non-positive load trend. Suppose node $N_i$ will work at a rate near to the maximum throughput at time $t + 1$ after receiving fragment $F_j$. We denote the resulted load trend of $N_i$ by $\hat{G}_{N_i,t}$, i.e. $\hat{G}_{N_i,t} = \hat{G}_{N_i,t} + \hat{G}_{F_j,t}$. If $\hat{G}_{N_i,t} > 0$, there will be a high chance that $N_i$ becomes overloaded within the time interval $t + 2$ to $t + \omega$. Therefore, a non-positive load trend is preferred, and we select destination node $N_i$ for fragment $F_j$ based on the following constraint:
5. ALGORITHM

In this section, details of the proposed algorithm are mentioned. We assume that the algorithm is hourly based and an hourly time series model (a seasonal ARIMA model with 24 hours as the seasonal period) is built for each fragment.

5.1 DATA REALLOCATION

The algorithm consists of 4 phases. In Phase 1, k-step ahead forecasting is performed to estimate the workloads and load trends in the coming hours. In Phase 2, node number adjustment is performed to add extra nodes or remove excessive nodes. In Phase 3, fragments with low migration costs are selected for indispensable migrations. In Phase 4, fragment migrations are performed for load balancing. Since the details of estimating workloads and load trends were described in the previous section, we skip Phase 1 and elaborate Phases 2-4 in the following paragraphs.

We outline the node number adjustment phase (Phase 2) in Algorithm 1. The algorithm first estimates the required number of working nodes for the coming ω time points */

<table>
<thead>
<tr>
<th>Algorithm 1. Node number adjustment phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: /* estimate the required number of working nodes for the coming ω time points */</td>
</tr>
<tr>
<td>2: for k = 1; k ≤ ω do</td>
</tr>
<tr>
<td>3: $\tilde{n}<em>{t+k} \leftarrow \sum</em>{j=t}^{t+\omega} \tilde{W}_{F_j,t}(k)/\theta$;</td>
</tr>
<tr>
<td>4: end for</td>
</tr>
<tr>
<td>5: if $n &lt; \tilde{n}_{t+1}$ then</td>
</tr>
<tr>
<td>6: $n \leftarrow \tilde{n}_{t+1}$;</td>
</tr>
<tr>
<td>7: Add ($\tilde{n}_{t+1} - n$) nodes;</td>
</tr>
<tr>
<td>8: else if $n &gt; \max_{1≤k≤\omega}(\tilde{n}_{t+k})$ then</td>
</tr>
<tr>
<td>9: $n_d \leftarrow \max_{1≤k≤\omega}(\tilde{n}_{t+k})$;</td>
</tr>
<tr>
<td>10: while $n_d &lt; n$ do</td>
</tr>
<tr>
<td>11: /* select the node with the lowest estimated workload at the coming time point */</td>
</tr>
<tr>
<td>12: $N_r \leftarrow {N_i; N_i \in \Omega_N, \exists N_j \in \Omega_N$ such that $\tilde{W}<em>{N_j,t}(1) &gt; \tilde{W}</em>{N_i,t}(1)}$;</td>
</tr>
<tr>
<td>13: /* add the fragments belonged to node $N_r$ to the migration set $\Omega_M$ */</td>
</tr>
<tr>
<td>14: for each $F_j \in \Omega_P$ such that $a_{F_j} = 1$ do</td>
</tr>
<tr>
<td>15: $\Omega_M \leftarrow \Omega_M \cup {F_j}$;</td>
</tr>
<tr>
<td>16: $a_{F_j} \leftarrow 0$;</td>
</tr>
<tr>
<td>17: end for</td>
</tr>
<tr>
<td>18: $n \leftarrow n - 1$;</td>
</tr>
<tr>
<td>19: $\Omega_N \leftarrow \Omega_N \setminus {N_r}$;</td>
</tr>
<tr>
<td>20: end while</td>
</tr>
<tr>
<td>21: end if</td>
</tr>
</tbody>
</table>

5.2 ALGORITHM 2. Fragment selection phase

<table>
<thead>
<tr>
<th>Algorithm 2. Fragment selection phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: for each $N_i \in \Omega_N$ do</td>
</tr>
<tr>
<td>2: while $\tilde{W}_{N_i,t}(1) &gt; \theta_u$ do</td>
</tr>
<tr>
<td>3: /* fragments belonged to node $N_i$ and satisfied Constraint 1 */</td>
</tr>
<tr>
<td>4: let $\Omega_r = {F_j; F_j \in \Omega_P, a_{i,j} = 1$ such that $</td>
</tr>
<tr>
<td>5: /* select the fragment with the lowest estimated workload at the coming time point */</td>
</tr>
<tr>
<td>6: if $\Omega_r \neq \emptyset$ then</td>
</tr>
<tr>
<td>7: $F_r \leftarrow {F_j; F_j \in \Omega_r, \exists F_i \in \Omega_r$ such that $\tilde{W}<em>{F_i,t}(1) &gt; \tilde{W}</em>{F_r,t}(1)}$;</td>
</tr>
<tr>
<td>8: else</td>
</tr>
<tr>
<td>9: $F_r \leftarrow {F_j; F_j \in \Omega_P, a_{i,j} = 1, \exists F_i \in \Omega_P, a_{i,i} = 1$ such that $\tilde{W}<em>{F_i,t}(1) &gt; \tilde{W}</em>{F_r,t}(1)}$;</td>
</tr>
<tr>
<td>10: end if</td>
</tr>
<tr>
<td>11: /* update the estimated workload and load trend of node $N_i$ after removing fragment $F_r$ */</td>
</tr>
<tr>
<td>12: $\tilde{W}<em>{N_i,t}(1) \leftarrow \tilde{W}</em>{N_i,t}(1) - \tilde{W}_{F_r,t}(1)$;</td>
</tr>
<tr>
<td>13: /* add fragment $F_r$ to the migration set $\Omega_M$ */</td>
</tr>
<tr>
<td>14: $\Omega_M \leftarrow \Omega_M \cup {F_r}$;</td>
</tr>
<tr>
<td>15: end while</td>
</tr>
<tr>
<td>16: end for</td>
</tr>
</tbody>
</table>

We outline the node number adjustment phase (Phase 2) in Algorithm 1. The algorithm first estimates the required numbers of working nodes for the coming ω time points, i.e. $\tilde{n}_{t+1}, \tilde{n}_{t+2}, \ldots, \tilde{n}_{t+ω}$ (Algorithm 1. Steps 2-4). New nodes are added if existing working nodes are not enough for the next time point, i.e. $n < \tilde{n}_{t+1}$ (Algorithm 1. Steps 5-7). Some nodes are removed if there are excessive working nodes for all ω time points, i.e. $n > \max_{1≤k≤\omega}(\tilde{n}_{t+k})$ (Algorithm 1. Step 8). In case of adding nodes, only the workloads in the coming time point are considered as to ensure there are sufficient working nodes for the coming time point. But for removing nodes, all ω time points are taken into account as to ensure no unnecessary migration cost will be generated within the next ω time points.

Suppose some nodes have to be removed. The algorithm iteratively selects a node with the lowest workload at time $t + 1$ for removal (Algorithm 1. Steps 9-19). From Das et al. (2011), migrating a fragment takes several seconds to several minutes. Since the migration of any fragment can be completed within a unit of time (an hour), we denote the migration cost of a fragment by its workload at time $t + 1$, and minimize the overall migration cost by selecting nodes with low workloads at time point $t + 1$ for removal.

After the node number adjustment phase, the algorithm checks if there are nodes that are going to be overloaded in the coming time point. If the nodes
**Algorithm 3. Fragment migration phase**

1: sort $F_j \in \Omega_M$ in descending order of $\bar{W}_{F_j,t}(1)$;
2: while $\Omega_M \neq \emptyset$ do
3: let $F_i$ be $F_j \in \Omega_M$ with the lowest $\bar{W}_{F_i,t}(1)$;
4: /* nodes that will not become overloaded after receiving fragment $F_i$ */
5: let $\Omega_L = \{N_i; N_i \in \Omega_S\}$ such that $\bar{W}_{N_i,t}(1) + \bar{W}_{F_i,t}(1) \leq \theta_u$;
6: /* nodes belonged to $\Omega_L$ and satisfied Constraint 2 */
7: let $\Omega_R = \{N_i; N_i \in \Omega_S\}$ such that $\bar{W}_{N_i,t}(1) + \bar{W}_{F_i,t}(1) \leq \theta_u$;
8: /* select the node with the lowest estimated workload at the coming time point */
9: if $\Omega_T \neq \emptyset$ then
10: $N_a = \{N_i; N_i \in \Omega_T, \forall N_i \in \Omega_T\}$ such that $\bar{W}_{N_a,t}(1) + \bar{W}_{N_i,t}(1)$;
11: else
12: $N_a = \{N_i; N_i \in \Omega_S, \forall N_i \in \Omega_S\}$ such that $\bar{W}_{N_a,t}(1) + \bar{W}_{N_i,t}(1)$;
13: end if
14: $\Omega_M \leftarrow \Omega_M - \{F_i\}$;
15: $a_{dr} \leftarrow 1$;
16: Migrate $F_i$ to $N_a$;
17: end while
18: remove all empty nodes;

exist, some fragments belonged to the nodes are selected for migrations. We outline the fragment selection phase (Phase 3) in Algorithm 2. Suppose a node will work at a rate greater than the upper bound at time $t + 1$, i.e. $\bar{W}_{N_i,t}(1) > \theta_u$. The algorithm iteratively selects a fragment to be removed from the node until the workload drops below $\theta_u$. If possible, the selected fragments have to satisfy Constraint 1 (See Section 4.3) (Algorithm 2, Steps 4-7) as to stabilize the node after removing fragments from the node. In case no fragment satisfying Constraint 1 is found, the algorithm applies the general rule, that is, selecting fragments with low workloads at time $t + 1$ (Alg. 2 Step 9) as to reduce the migration cost.

Finally, fragment migrations are performed. We outline the fragment migration phase (Phase 4) in Algorithm 3. Suppose some fragments have to be migrated from their original nodes. After sorting the fragments in descending order of their estimated workloads at time $t + 1$ (Algorithm 3, Step 1), the algorithm iteratively selects a destination node for each fragment. To avoid generating unnecessary migration costs, the selected nodes are checked not to become overloaded after receiving the fragments, i.e. $\bar{W}_{N_i,t}(1) + \bar{W}_{F_i,t}(1) \leq \theta_u$ (Algorithm 3, Step 5). If possible, the selected nodes also have to satisfy Constraint 2 (See Section 4.3) (Algorithm 3, Steps 7-10) as to further reduce the chance of generating unnecessary migration costs.

### 5.2 Correctness

We prove the correctness of the proposed algorithm under the assumption that workloads can be modeled as observed time series, and show that, with accurate forecastings, the proposed algorithm correctly adjusts the number of working nodes and reallocates fragments.

For the proposed algorithm, if it is found that a node has a high chance of being overloaded at the coming time point, fragments belonged to the node will be removed during the fragment selection phase until the estimated workload at the next time point drops below $\theta_u$. Therefore, to prove that there is no overloaded node, we only have to show that no node will work at a rate greater than $\theta_u$ after receiving fragments during the fragment migration phase. In other words, we have to guarantee that there always exists a node $N_i$ for receiving the migrated fragment $F_j$ such that $\bar{W}_{N_i,t}(1) + \bar{W}_{F_j,t}(1) \leq \theta_u$.

**Lemma 2.** Given that $\theta_u - \theta_1 \geq W_{F_j,t}$ for any fragment $F_j$ at any time $t$. During the fragment migration phase, there exists a node $N_i$ for receiving fragment $F_j$ such that $\bar{W}_{N_i,t}(1) + \bar{W}_{F_j,t}(1) \leq \theta_u$.

**Proof.** We prove by contradiction. Suppose the algorithm is looking for a node for receiving fragment $F_i$ during the fragment migration phase but does not exist a node $N_i$ such that $\bar{W}_{N_i,t}(1) + \bar{W}_{F_i,t}(1) \leq \theta_u$. For any node $N_i$,

$$\bar{W}_{N_i,t}(1) + \bar{W}_{F_i,t}(1) > \theta_u$$

Consequently, the sum of the workloads of all nodes is greater than multiplying $\theta_1$ by the number of working nodes $n$.

$$\sum_{i=1}^{n} \bar{W}_{N_i,t}(1) > \theta_1 \times n$$

However, according to Algorithm 1, after the node number adjustment phase, the number of working nodes $n \geq \bar{n}_{t+1}$. Namely,

$$n \geq \sum_{j=1}^{m} \bar{W}_{F_j,t}(1)/\theta_1$$
As mentioned in Section 4.1, under two scenarios, unnecessary migration costs will be generated. In Section 4.3, we described that, by applying Constraint 2, unnecessary migration costs generated from fragment migrations can be reduced. Here we further show that unnecessary migration costs resulted from node removal can be reduced, that is, no node will be removed if some nodes have to be added back within the time interval \( t + 1 \) to \( t + \omega \).

**Lemma 3.** Suppose a node is removed at time \( t \). No node is intended to be added back within the time interval \( t + 1 \) to \( t + \omega \).

**Proof.** We prove by contradiction. Suppose a node is removed at time \( t \) but some nodes are intended to be added back with the time interval \( t + 1 \) to \( t + \omega \). Before removing the node, the number of working nodes \( n \) should be less than or equal to the estimated numbers of working nodes for the coming \( \omega \) point times, i.e. \( n \leq \max_{s \in S \cap \{t+1\}} (\bar{\lambda}_{t+k}) \). However, according to Algorithm 1, node removal will be performed only when \( n > \max_{s \in S \cap \{t+1\}} (\bar{\lambda}_{t+k}) \) (Algorithm 1, Steps 8-20). It is contradicted. ■

**Theorem 1.** The proposed algorithm reallocates fragments and adjusts the number of working nodes at any time \( t \) with minimum migration costs such that there is no overloaded or excessive working node at time \( t + 1 \).

**Proof.** In Lemma 2, we proved that no node will work at a rate greater than the upper bound \( \theta_U \) after receiving fragments during the fragment migration phase. In Algorithm 1, it is explicitly shown that excessive working nodes will be removed in the case that no additional migration cost will be resulted in the near future. Hence, we can argue that there is no overloaded or excessive working node at time \( t + 1 \) after performing the algorithm at time \( t \), and what remains to be proved is that the migration cost is minimized.

To simplify the proof, we consider the cost minimization problem as an optimization problem and express the optimum solution of the problem as the combination of optimum solutions of its subproblems (Cormen et al., 2001). Namely, to prove that the overall migration cost is minimized, we have to show that the migration cost generated from each single step is minimized. For the proposed algorithm, migration costs will be generated in two situations, when excessive nodes have to be removed and when fragments have to be migrated from overloaded nodes. For the former situation, the proposed algorithm iteratively selects a node with the lowest workload at time \( t + 1 \) for removal (Algorithm 1, Steps 10-19). For the latter situation, the proposed algorithm iteratively selects a fragment with the lowest workload at time \( t + 1 \) for migration (Algorithm 2, Step 9). Clearly, in both situations, the migration cost generated in each iteration is minimized. ■

In Section 1, we claimed that the proposed algorithm is a generalization of threshold-based algorithms. Here we prove the claim by showing that the proposed algorithm will give the same functionality when the best time series model for representing fragment workloads is an ARIMA(1,0,0) model with \( \phi_1 = 1 \).

**Lemma 4.** The proposed algorithm will be reduced to a threshold-based algorithm when fragment workloads are modeled as an ARIMA(1,0,0) model with \( \phi_1 = 1 \).

**Proof.** Suppose the best time series model for representing fragment workloads is an ARIMA(1,0,0) model with \( \phi_1 = 1 \). For any fragment \( F_j \), the workload at time \( t \) is expressed as:

\[
W_{F_j,t} = W_{F_j,t-1} + a_t
\]

The workloads for the coming \( \omega \) time points are estimated as follows:

\[
\bar{\omega}_{F_j,t}(1) = W_{F_j,t}
\]

\[
\bar{\omega}_{F_j,t}(2) = \bar{\omega}_{F_j,t}(1)
\]

\[
\vdots
\]

\[
\bar{\omega}_{F_j,t}(\omega) = \bar{\omega}_{F_j,t}(\omega-1)
\]

At any time \( t \), the estimated workloads for the coming \( \omega \) time points are the same and equal to the measured workload at time \( t \). Therefore, during the node number adjustment phase, we can regard the proposed algorithm adds or removes nodes simply based on the current workload to the database system since there is no difference between taking the current workloads or the estimated future workloads as the adjustment criteria. It is the same as a threshold-based algorithm that performs actions based on the current workloads.

Similarly, during the fragment selection phase and the fragment migration phase, the proposed algorithm also performs actions like a threshold-based
Algorithm. Since the workloads for the coming \( \omega \) time points are the same, at any time \( t \), the estimated load trends of all fragments and all nodes become 0. Constraint 1 and Constraint 2 (See Section 4.3) are always satisfied and can be neglected. Therefore, with the same reason, we can regard that the proposed algorithm performs fragment reallocation simply based on the current workloads.

6. EVALUATION

There are two experiments. The first experiment analyzes real access logs and evaluates the accuracy of ARIMA models in load forecasting. The second experiment simulates the dynamic changing in a scalable distributed database system and compares the proposed algorithm with a threshold-based algorithm.

6.1 TIME SERIES ANALYSIS

For the proposed algorithm, the performance is highly dependent on the accuracy of time series forecasting. Therefore, we analyze real access logs in the first experiment in order to show that observed time series can be found in the workload of an online application. The access logs we used are from 1998 World Cup Website (Arlitt & Jin, 1998). The access logs include all requests made to the website from April 30, 1998 to July 26, 1998.

In the analysis, we use the hourly counts from May 1, 1998 to May 31, 1998 as the training set for building the ARIMA model. The model and related parameters are generated by a statistical software called R (R Development Core Team, 2011). To see if there is a daily factor in the model, we set 24 hours as the seasonal period. From the output of R, an ARIMA(1,1,1) × (2,0,2)\(_{24}\) model is found and the coefficients are shown in Table 3. Compare the orders and the coefficients, we find that the seasonal AR part contributes most to the series. It implies that there is truly a daily factor in the model.

In the next step, we verify the correctness of the model by performing \( k \)-step ahead forecasting. We take the hourly counts from June 1, 1998 to June 7, 1998 as the validation set. There are totally 168 sample data points (See Figure 3(a)). For each data point, we perform 1-step ahead forecasting to estimate the number of accesses in the next hour. From the result, we find that the estimated values are very close to the exact values (See Figure 3(b)) and the percentage error is around 5%. Similarly, we perform forecasting up to 5-step ahead for each data point. From the results, we find that the estimated values become less accurate (See Figure 3(c)-3(f)). But still, the change in load trend is clearly shown.

<table>
<thead>
<tr>
<th>AR1(( \phi_1 ))</th>
<th>MA1(( \psi_1 ))</th>
<th>SAR1(( \Phi_1 ))</th>
<th>SAR2(( \Phi_2 ))</th>
<th>SMA1(( \Psi_1 ))</th>
<th>SMA2(( \Psi_2 ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0368</td>
<td>-0.5244</td>
<td>0.6645</td>
<td>0.3928</td>
<td>-0.5288</td>
<td>-0.4424</td>
</tr>
<tr>
<td>0.0692</td>
<td>0.0580</td>
<td>0.1805</td>
<td>0.1854</td>
<td>0.1623</td>
<td>0.1576</td>
</tr>
</tbody>
</table>

Table 3. Coefficients of the ARIMA(1,1,1) × (2,0,2)\(_{24}\) model

Figure 3. Hourly access logs and \( k \)-step ahead forecasting results
6.2 Simulation

Due to the lack of real data, in the second experiment, synthetic workloads are used to simulate the dynamic changing in a scalable distributed database system. We assume workloads can be modeled as observed time series and generate fragment workloads by the following seasonal ARIMA model:

\[ W_{F,j,t} = \phi_1 W_{F,j,t-1} + \Phi_1 W_{F,j,t-24} - \phi_1 \Phi_1 W_{F,j,t-25} + a_t \]

\( \phi_1 \) and \( \Phi_1 \) are coefficients, which are unique for each series of fragment workloads. \( \{a_t\} \) is a series of white noise error terms generated from \( R \). We restrict the mean values of \( \{W_{F,j,t}\} \) to be around 100 and introduce different levels of noise into \( \{W_{F,j,t}\} \) by changing the variance of \( \{a_t\} \). There are three sets of synthetic data generated. Each set consists of 100 series and each series contains 96 data points representing fragment workloads in four days. The variances of \( \{a_t\} \) used for generating the datasets with low, moderate and high levels of noise are 5, 15 and 30 respectively.

In the simulation, we assume that each node can handle maximally 1800 requests in a unit of time and the cost for migrating a fragment at a particular time point is the workload of the fragment at that time point since the workload of a fragment at a particular time point shows, on average, how many requests will be suspended during the migration of the fragment at that time point. With the goal of keeping all nodes working at a rate around 95% throughput, the proposed algorithm is compared with a simple threshold-based algorithm, which performs actions based on the workloads measured at the current time point. For the proposed algorithm, we set the lower bound \( \theta_l \) and the upper bound \( \theta_u \) by \( \pm 2.5\% \) to the expected working rate. Forecasting is performed up to 5-step. For the threshold-based algorithm, we set the threshold value to be \( \theta_u \) for the purpose of comparison.

Figures 4-6 show the simulation results of using the proposed algorithm and the threshold-based algorithm to perform node number adjustment and fragment reallocation. For all datasets, the proposed algorithm gives a better performance. It is rare to have node overloading (See Figures 4(a)-6(a)). The migration cost generated is lower than that of the threshold-based algorithm (See Figures 4(b)-6(b)). When there are suddenly drops in the workload, the proposed algorithm does not remove excessive nodes (See Figures 4(c)-6(c)), and therefore no unnecessary migration cost is generated.

7. RELATED WORK

To the best of our knowledge, there is no previous work formally addressing the problem of data allocation in scalable distributed database systems. But still, our work is related to data allocation algorithms for traditional distributed database systems as well as the design of scalable database system.

7.1 Data Allocation in Traditional Distributed Database Systems
The problem of data allocation in traditional distributed database system was defined by Apers (1988). It is a variation of file allocation problem (Chu, 1969) in which access patterns are assumed to be static. Algorithms were proposed for finding optimal allocation schema (Huang & Chen, 2001; Ahmad et al., 2002; Menon 2005). Since access patterns of real applications are unlikely to be static, Brunstrom et al. (1995) proposed a dynamic data allocation algorithm, which reallocates data fragment when there is a change in access patterns. Similar threshold-based algorithms for reallocating non-replicated fragments were kept proposing (Ulus & Uysal, 2003; Singh & Kahlon, 2009). On the contrary, Wolfson et al. (1997) proposed an algorithm for dynamic replication of a fragment. The algorithm aims at moving the replication schema towards an optimal one.

7.2 SCALABLE DATABASE DESIGN

Das et al. (2010) used a threshold-based algorithm to reallocate data fragments in their prototype system. Curino et al. (2011) used non-linear optimization techniques to reallocate resource in their database systems. An engine for monitoring and consolidation was developed and published in the same year. The engine measures the hardware requirements of database workloads and predicts the resource utilization such that resource allocation can be performed accurately. Soundararajan et al. (2009) introduced a multi-resource allocator to dynamically allocate resources for database servers running on virtual storage, which give an idea on how to configure the resources in an elastic environment.

7.3 LIVE MIGRATION TECHNIQUES

To lighten the influence due to data migration, live migration techniques are required. Clark et al. (2005) proposed techniques for migrating operating system instances across distinct physical hosts with minimal service downtimes. Das et al. (2011) proposed techniques for live database migration in shared storage architecture. Elmore et al. (2011) proposed techniques for live database migration in shared nothing architecture. Barker et al. (2012) proposed an end-to-end database migration system that works at the middleware level. However, from these studies, we see that migration costs still cannot be neglected, and therefore we have a motivation on designing data allocation algorithm that minimizes performance degradation resulted from fragment migrations.

8. CONCLUSION

In this paper, we defined the problem of data allocation in scalable distributed database systems and presented an efficient algorithm, which makes use of time series models to perform node number adjustment and fragment reallocation. From the simulation, we saw that data allocation is performed in a reasonable way. Load balancing and resource-saving can be achieved under the assumption that future workloads can be modeled as observed time series. With accurate forecastings, the performance of the proposed algorithm is much better than that of a threshold-based algorithm.

9. REFERENCES


Authors

Shun-Pun Li received his B.Sc. degree in Computer Science with a minor in Statistics from the Chinese University of Hong Kong in 2011. He is currently an M.Phil. student in the Department of Computer Science and Engineering at the same university. His research interests include data privacy and distributed systems.

Man-Hon Wong received his B.Sc. and M.Phil. degrees from the Chinese University of Hong Kong in 1987 and 1989 respectively. He then went to University of California at Santa Barbara where he got the Ph.D. degree in 1993. He joined The Chinese University of Hong Kong in 1993 as an assistant professor. He was promoted to associate professor in 1998. His research interests include transaction management, mobile databases, data replication, distributed systems, expert systems and applications of fuzzy logic.
Call for Articles

International Journal of Big Data
ISSN: 2326-442X     http://www.hipore.com/ijbd/

Mission
Big Data has become a valuable resource and mechanism for the practitioners and researchers to explore the value of data sets in all kinds of business scenarios and scientific investigations. New computing platforms such as cloud computing, mobile Internet, social network are driving the innovations of big data. From government initiative perspective, Obama Administration in United States launched "Big Data" initiative that announces $200 Million in new R&D investments on March 29, 2012. European Union also announced "Big Data at your service" on July 25, 2012. 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 big data.

The International Journal of Big Data (IJBD) aims to provide a reputable publication channel for all authors working in the field of all aspects of Big Data. Big Data is a dynamic discipline. One of the objectives of IJBD is to promote research accomplishments and new directions. Therefore, IJBD welcomes special issues in any emerging areas of big data.

Topics
IJBD includes topics related to the advancements in the state of the art standards and practices of Big Data as well as emerging research topics defining the future of Big Data. Topics of interest to include, but are not limited to, the following:

Big Data Models and Algorithms (Foundational Models for Big Data, Algorithms and Programming Techniques for Big Data Processing, Big Data Analytics and Metrics, Representation Formats for Multimedia Big Data)

Big Data Architectures (Cloud Computing Techniques for Big Data, Big Data as a Service, Big Data Open Platforms, Big Data in Mobile and Pervasive Computing)

Big Data Management (Big Data Persistence and Preservation, Big Data Quality and Provenance Control, Management Issues of Social Network enabled Big Data)

Big Data Protection, Integrity and Privacy (Models and Languages for Big Data Protection, Privacy Preserving Big Data Analytics Big Data Encryption)

Security Applications of Big Data (Anomaly Detection in Very Large Scale Systems, Collaborative Threat Detection using Big Data Analytics)

Big Data Search and Mining (Algorithms and Systems for Big Data Search, Distributed, and Peer-to-peer Search, Machine learning based on Big Data, Visualization Analytics for Big Data)
