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.
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 forecasting.
- 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 its 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 \( F_i \) is owned by node \( N_j \); 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 \( N_4 \) as \( a_{11} = 1 \) and \( a_{33} = 1 \). Fragments \( F_2 \) and \( F_5 \) are owned by node \( N_5 \) 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 \( N_4 \). Therefore, \( W_{N_i,1} = W_{F_1,1} + W_{F_3,1} = 5 \).

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 overloading, 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 \( F_5 \) 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 overloading, migration costs may result from increasing or decreasing the number of working nodes. We use generalization to 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, \( F_3 \) 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 relocation 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.

Scenarios 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 \( N_2 \).

Scenarios 2. Suppose the maximum throughput of a node is 10. At time \( t = 2 \), the workload to the system increases to 23. A node has to be added back and fragments need to be reallocated.

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.

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

\( a_t, a_{t-1}, ... \) 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:

\[
\phi(B) = 1 - \phi_1 B - ... - \phi_p B^p
\]

\[
\psi(B) = 1 - \psi_1 B - ... - \psi_q B^q
\]

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

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

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

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

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:

$$
\hat{W}_{F_1,2}(1) = 0.4W_{F_1,2} + 2W_{F_1,1} = 4.8 \\
\hat{W}_{F_1,2}(2) = 0.4\hat{W}_{F_1,1} + 2W_{F_1,2} = 5.92
$$

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 $\tilde{\omega}_{F_j,t}$, 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:

$$
\tilde{\omega}_{F_j,t} = \frac{\sum_{t=1}^{\omega} \left( t - \frac{1}{\omega} \sum_{i=1}^{\omega} v \right) \left( \hat{W}_{F_j,t}(v) - \frac{1}{\omega} \sum_{i=1}^{\omega} \hat{W}_{F_j,t}(v) \right)}{\sum_{t=1}^{\omega} \left( t - \frac{1}{\omega} \sum_{i=1}^{\omega} v \right)^2}
$$

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 $F_1, F_2, \ldots, F_u$.

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

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

$$
= \sum_{j=1}^{u} \left( t - \frac{1}{\omega} \sum_{i=1}^{\omega} v \right)^2
$$
We say node $N_i$ is more stable than node $N_l$ if $|\hat{G}_{N_l,t}| < |\hat{G}_{N_i,t}|$. In other words, a node is stable if its load trend is close to zero.

**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.

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:

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

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 $N_l$ for fragment $F_j$ based on the following constraint:

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

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:

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

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 $\theta_l$ as the denominator because we are less willing to see a node being overloaded than underloaded.

In practice, the values of $\theta_u$ and $\theta_l$ can be determined by queueing theory (Ng & Soong, 2008). 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 $[\rho_u, \rho_l]$ by $F$-test (Lilliefors, 1966). Then, $\theta_u$ and $\theta_l$ are obtained as follows:

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

<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>

$\hat{G}_{N_i,t}$

Table 2. Workloads under different allocation schemas
Algorithm 1. Node number adjustment phase

1: /* estimate the required number of working nodes for the coming \( \omega \) time points */
2: for \( k = 1: \omega \) do
3: \( \tilde{n}_{t+k} \leftarrow \frac{\sum_{j=1}^{m} \tilde{W}_{F_i,t}(k)}{\theta_k} \);
4: end for
5: if \( n < \tilde{n}_{t+1} \) then
6: \( n \leftarrow \tilde{n}_{t+1} \);
7: Add \((\tilde{n}_{t+1} - n)\) nodes;
8: else if \( n > \max_{1 \leq k \leq \omega}(\tilde{n}_{t+k}) \) then
9: \( n_d \leftarrow \max_{1 \leq k \leq \omega}(\tilde{n}_{t+k}) \);
10: while \( n_d < n \) do
11: /* select the node with the lowest estimated workload at the coming time point */
12: \( \Omega_r \leftarrow \{N_i; N_i \in \Omega_N, \exists F_j \in \Omega_F \}
13: /* add the fragments belonged to node \( N_r \) to the migration set \( \Omega_M \) */
14: for each \( F_j \in \Omega_F \) such that \( a_{F_j} = 1 \) do
15: \( \Omega_M \leftarrow \Omega_M \cup \{F_j\} \);
16: \( a_{F_j} \leftarrow 0 \);
17: end for
18: \( n \leftarrow n - 1 \);
19: \( \Omega_N \leftarrow \Omega_N - \{N_r\} \);
20: end while
21: end if

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 numbers of working nodes for the coming \( \omega \) time points, i.e., \( \tilde{n}_{t+1}, \tilde{n}_{t+2}, \ldots, \tilde{n}_{t+\omega} \) (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 \( \omega \) time points, i.e. \( n > \max_{1 \leq k \leq \omega}(\tilde{n}_{t+k}) \) (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 \( \omega \) time points are taken into account as to ensure no unnecessary migration cost will be generated within the next \( \omega \) 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 $F_i \in \Omega_M$ in descending order of $\bar{W}_{F_i,t}(1)$;
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 $\Omega_S = \{N_i; N_i \in \Omega_S$
   such that $\bar{W}_{N_i,t}(1) + \bar{W}_{F_i,t}(1) \leq \theta_u\}$;
6: /* nodes belong to $\Omega_S$ and satisfied Constraint 2 */
7: let $\Omega_F = \{N_i; N_i \in \Omega_S$
   such that $\bar{C}_{N_i,t} + \bar{C}_{F_i,t} \leq \theta_u\}$;
8: /* select the node with the lowest estimated workload at the coming time point */
9: if $\Omega_F \neq \emptyset$ then
10: $N_d \leftarrow \{N_i; N_i \in \Omega_F, \exists N_j \in \Omega_F$
   such that $\bar{W}_{N_j,t}(1) > \bar{W}_{N_i,t}(1)\}$;
11: else
12: $N_d \leftarrow \{N_i; N_i \in \Omega_S, \exists N_j \in \Omega_S$
   such that $\bar{W}_{N_i,t}(1) > \bar{W}_{N_j,t}(1)\}$;
13: end if
14: $\Omega_M \leftarrow \Omega_M - \{F_i\}$;
15: $\alpha_{dr} \leftarrow 1$;
16: Migrate $F_i$ to $N_d$;
17: end while
18: remove all empty nodes;

exist, some fragments belong 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 forecasting, 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 belong 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 $\theta_u - \theta_t \geq \bar{W}_{F_j,t}$ 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 $F_j$ during the fragment migration phase but there does not exist a node $N_i$ such that $\bar{W}_{N_i,t}(1) + \bar{W}_{F_j,t} \leq \theta_u$. For any node $N_i$,

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

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

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

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

$$n \geq \sum_{j=1}^{m} \bar{W}_{F_j,t}(1)/\theta_t$$
\[ n \times \theta_t \geq \sum_{i=1}^{n} \bar{W}_{N,t}(1) \]

It is contradicted. ■

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. \( n \leq \max_{s \in S} (\bar{n}_{t+k}) \). However, according to Algorithm 1, node removal will be performed only when \( n > \max_{s \in S} (\bar{n}_{t+k}) \) (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 \( \theta_n \) 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:

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

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

<table>
<thead>
<tr>
<th></th>
<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>Coefficient</td>
<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>Standard error</td>
<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>

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.
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:

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

\(\phi_1, \Phi_1\) 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 over loadings (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 system 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.