SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

September 2012 Vol. 55 No. 9: 1935–1948
doi: 10.1007/s11432-011-4432-3

MBA: A market-based approach to data allocation and dynamic migration for cloud database

WANG TengJiao[1]*, LIN ZiYu[2], YANG BiShan[1], GAO Jun[1], HUANG Allen[3], YANG DongQing[1], ZHANG Qi[1], TANG ShiWei[1] & NIU JinZhong[4]

[1]*Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education of China; School of Electronics Engineering and Computer Science, Peking University, Beijing* 100871*, China;*
[2]*School of Information Science and Technology, Xiamen University, Xiamen* 361005*, China;*
[3]*Microsoft SQL China R&D Center, Beijing* 100080*, China;*
[4]*Computer Science, Graduate School and University Center, City University of New York, New York NY* 10016*, USA*

Received October 1, 2011; accepted December 12, 2011; published online March 12, 2012

Abstract With the coming shift to cloud computing, cloud database is emerging to provide database service over the Internet. In the cloud-based environment, data are distributed at Internet scale and the system needs to handle a huge number of user queries simultaneously without delay. How data are distributed among the servers has a crucial impact on the query load distribution and the system response time. In this paper, we propose a market-based control method, called MBA, to achieve query load balance via reasonable data distribution. In MBA, database nodes are treated as traders in a market, and certain market rules are used to intelligently decide data allocation and migration. We built a prototype system and conducted extensive experiments. Experimental results show that the MBA method significantly improves system performance in terms of average query response time and fairness.

Keywords market-based control, data distribution, load balancing

Citation Wang T J, Lin Z Y, Yang B S, et al. MBA: A market-based approach to data allocation and dynamic migration for cloud database. Sci China Inf Sci, 2012, 55: 1935–1948, doi: 10.1007/s11432-011-4432-3

1 Introduction

Cloud computing is a fast-developing computing paradigm where data are distributed at massive scales and services are available for users over the Internet [1]. The currently available "Clouds", like Amazon's S3 and Google FS (GFS), typically provide unstructured storage services. Also, more and more modern large-scale applications have strong demand for database as a service over the Internet.

Cloud database is emerging to serve this end. Large database provider like Microsoft SQL puts much concern on the techniques about developing an effective cloud database. In the joint research project between Peking University and Microsoft Cloud DB Software Center, we are focusing on how to improve load balancing mechanism in cloud database, which is a critical issue for improving system's performance.

Generally, cloud database has the following features:

*Corresponding author (email: tjwang@pku.edu.cn)

• Fast response and high efficiency in serving queries are guaranteed. Importantly, load balancing should be maintained through reasonable data distribution, and thus favorable system performance can be guaranteed.

• The system can provide high manageability and availability with the ever-changing query patterns.

• The topology of database is dynamic; that is, database nodes may be added or deleted over time.

In the cloud database, how data are distributed in the system is a key factor which affects the load balancing resolution. Besides, the workload history on data and the workload changes in the system also play important roles in balancing query load.

In this paper, we developed an effective method MBA (market, bid and ask) to achieve reasonable data distribution among cloud DB nodes, and balance the workload and improve the overall DB operation performance in the cloud database. MBA is a novel market-based control method. In MBA, database nodes are treated as traders in a market, and certain market rules are used to intelligently decide data allocation and migration. With this technique, MBA can achieve load balancing very fast, and adapt to the ever-changing query load and database node topology effectively. Specifically, our method has the following contributions:

• Fast convergence to query load balance based on market mechanism. Database nodes are considered as traders in a market and interact with each other under certain market rules [2]. With the intelligent market mechanism, MBA can achieve equilibrium on the workload quickly via query load exchanging.

• Quickly adapting to query load evolution. In a cloud database system, query load distribution evolves with the change of user query patterns. MBA can quickly make the system converge to a new equilibrium state whenever the previous equilibrium is broken.

• Effectively accommodating to "cloud drift" with market mechanism. Cloud drift means the evolution of database node topology and data location over time. In fact, the change of database node topology can be seen as that of trader participation. Market mechanism, famous for being "free in and out", is good at accommodating itself to this scenario.

Moreover, a prototype system named MADAM (market-based data allocation and migration) is developed to carry out a series of experiments. We obtained customer data from a mobile communication company. Experimental results on the data set show that MBA significantly improves system performance in terms of average query response time and fairness.

The remainder of this paper is organized as follows: Section 2 discusses the related work; Section 3 introduces cloud database and problem statement; Section 4 describes an overview of market-based data allocation and migration; Section 5 presents in detail the MBA method; the experimental results are reported in Section 6; we finally give the discussion and conclusion in Section 7.

2 Related work

2.1 Load balancing

Many research have been done to address the issue of load balancing in distributed systems. They are typically classified into static and dynamic ones [3]. Static policies determine job assignments based on information about the average behavior of the system. They are simple, efficient, and easy to implement, but cannot adapt to the changing workload in the system. To respond to the fluctuation of workload over time, dynamic load balancing policies are proposed. Ref. [4], for example, gives a policy with a central job dispatcher, which periodically collects load information at system nodes and makes decisions for job transformation. They are more complex than static policies, but exhibit better performance. These load balancing policies view system nodes as a collection of common resources for arriving jobs. This makes them inapplicable to the scenario we are concerned about, in which a query should be processed by a particular server where the queried data are located. Refs. [5–7] address the issue of data allocation; however they are typically designed to allocate web documents among a cluster of web servers, where the document size and HTTP connection numbers are main concerns.

A more relative work is data distribution in parallel storage systems. In these systems, horizontal data partitioning has been commonly used to distribute data among system nodes. There are typically three

strategies for horizontal partitioning: round-robin, hash, and value-range partitioning. These partitioning strategies each has advantages and disadvantages, but they do not take into consideration the access pattern on data, which would be a problem when the pattern is not a uniform distribution. Dynamic data reallocation needs to effectively adapt to changes of access patterns. When access frequencies of data items lead to unbalanced workload on system nodes, data migration should be performed to balance workload. The simple data skew handling method in [8] balances the storage space for data, but does not guarantee balanced data access load across system nodes. Similarly most access load skew handling methods do not consider data balancing [9,10]. In [11], a data-placement method was proposed to balance both access load and storage space of data, but it is designed for parallel storage system, and focuses on achieving availability and scalability for parallel storage configuration.

2.2 Market mechanism

During the past few years, market mechanism has gained much interest in both the economics and computer science literature. Market-based techniques [2] are applying themselves to resource allocation, often through the creation of artificial market. This is due to the fact that market mechanisms, when well designed, can achieve desired economic outcomes like high allocated efficiency. Auctions are a subclass of market mechanisms that have received particular attention. Double auction [12], on which our method is based, is one type of auction where both buyers and sellers can make offers. There is a wide range of literature published on double auctions (e.g., [13,14]).

Connecting load balancing to the market mechanism in the distributed environment is a new direction of research. In our approach, we will show how to integrate the ideas of the artificial market into solving load balancing challenges in the cloud-based environment.

3 Problem statement

In this section we will first give an introduction to cloud database, and then the problem to be solved is defined formally.

As Figure 1 shows, a typical cloud database includes cloud database nodes (CDNs), cloud database master nodes (CDMNs) and application clients. In a cloud database, data are partitioned across CDNs. CDNs are distributed, heterogeneous database systems, with each storing a subset of records. CDMNs play the role of data nodes manager, and guide certain application clients to proper data nodes for data accessing. When a user sends a query through an application client, it first arrives at one of the CDMNs and then is routed to the CDN where the queried data are located. The CDN then executes the query on its local database, and returns the requested information.

Now we give some definitions related to query load in the cloud database.

Definition 1. The query load associated with a data unit R (i.e., a row) in a time interval Δt, denoted by l, is its read frequency during Δt. Δt is measured in terms of ten seconds, which is attractive in most applications.

Definition 2. Given a partition P with k data units and their respectively corresponding loads $l_1, l_2, \ldots, l_k$, the query load associated with P is expressed as $L_P = \sum_{j=1}^{k} l_j$.

Definition 3. Given a CDN D with m partitions and their respective corresponding loads $L_P^1, L_P^2, \ldots, L_P^m$, the query load associated with D is expressed as $L_D = \sum_{j=1}^{m} L_P^j$.

Suppose a cloud database D, which is composed of data nodes of D_1, $D_2, \ldots, D_N$, is with total processing capacities c_D and with query load vector $\boldsymbol{L}_D^0 = \{L_{D_1}, L_{D_2}, \ldots, L_{D_N}\}$, where L_{D_i} represents the query load on data node D_i, and $\sum_{i=1}^{N} L_{D_i} < c_D$. Let c_i be the processing capacity (or computing power) of data node D_i, and let r_i, the unused computing power of D_i, be defined as $c_i - L_{D_i}$. If $c_1 = c_2 = \cdots = c_N$, all CDNs have the same processing capacity and we call such cloud database a homogeneous one opposed to a heterogeneous one where various CDNs are with non-uniform processing capacities. In real applications, a cloud database is usually heterogeneous.

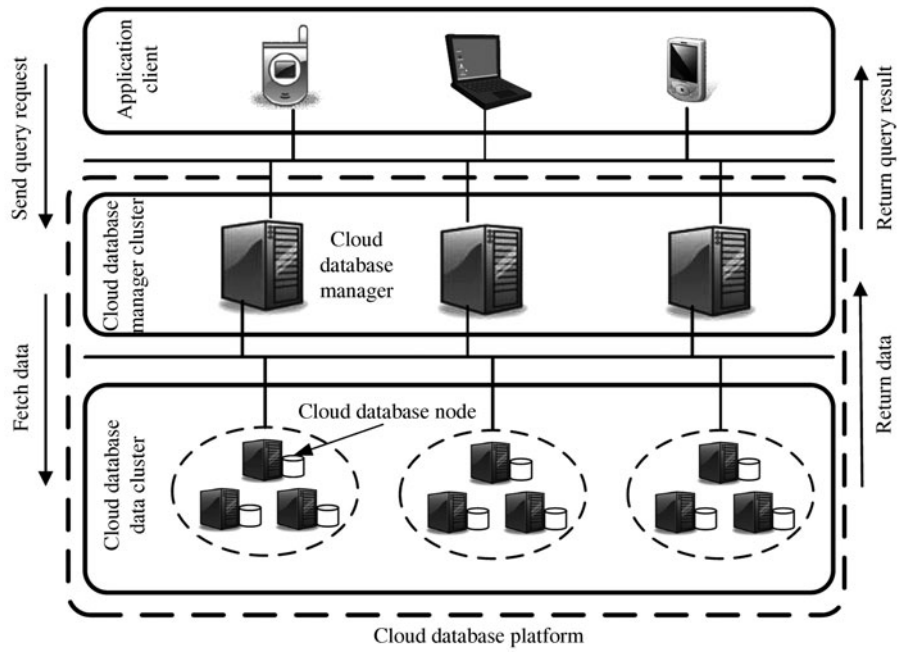

Figure 1 The model of cloud database.

Definition 4. Suppose a cloud database D with N data nodes satisfying $c_1 \leqslant c_2 \leqslant \cdots \leqslant c_i \leqslant \cdots \leqslant c_N$. If there exists a k $(1 \leqslant k \leqslant N)$ satisfying

1) $L_{D_1} = L_{D_2} = \cdots = L_{D_{k-1}} = 0$,
2) $L_{D_k}, L_{D_{k+1}}, \ldots, L_{D_N} \neq 0$,
3) $r_k = r_{k+1} = \cdots = r_N$ and
4) $r_k \geqslant c_{k-1}$,

then we say that D is query load balanced.

The condition of $r_k = r_{k+1} = \cdots = r_N$ means a strict balance state. In real applications, we can hardly make unused computing powers of various CDNs strictly equal to each other. The goal of MBA is to diminish the difference of unused powers on various CDNs as much as possible.

Suppose a cloud database D with a query load distribution of $\boldsymbol{L}_D^0 = \{L_{D_1}, L_{D_2}, \ldots, L_{D_N}\}$ satisfying $c_1 \geqslant c_2 \geqslant \cdots \geqslant c_i \geqslant \cdots \geqslant c_N$, and $\boldsymbol{L}_D^0$ is not in a query load balanced state. The problem of query load balancing for D is identified as a function $f(\boldsymbol{L}_D^0) \to \boldsymbol{L}_D^1$, where $\boldsymbol{L}_D^1 = \{L_{D_1}^*, L_{D_2}^*, \ldots, L_{D_N}^*\}$, and $\boldsymbol{L}_D^1$ is in a query load balanced state.

4 An overview of market-based data allocation and dynamic migration

Figure 2 illustrates the architecture of the MADAM system. The market-based data allocation and dynamic migration runs on the components with gray background (DTA and DAA). The other components are those necessary for the implementation of a cloud database system and here they are simply classified as CDSCs (cloud database system components). CDSCs include local partitioning map, global partitioning map, partition manager, name resolution and so on. In a cloud database system, there are usually more than one CDMNs (cloud database master nodes) for high availability. Therefore, there is no central CDMN, and some information such as global partition map is further replicated in all the CDMNs. Client applications can get services from any CDMN. For implicity, there is only one CDMN in Figure 2.

In MBA approach, there are two types of agents, i.e., DTA (data trading agent) and DAA (data auctioning agent), working together to achieve query load balancing. A DTA is responsible for the real-time tracking of query load on a CDN (cloud database node), and for determining the bid or ask price for the CDN it resides on. DAA plays the role of market institution, which defines how a transaction takes

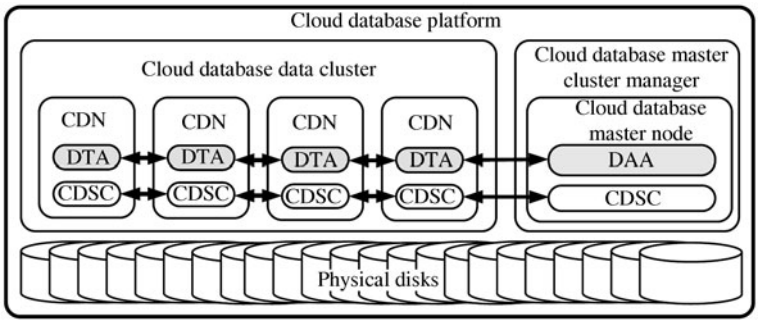

Figure 2 The architecture of MADAM.

place, in other words, which traders (i.e., CDNs) can buy and which traders can sell. Since there are more than one DAAs residing on various CDMNs, each DTA on ceratin CDN may make its own decision on joining in which market, i.e., being administered by which DAA. It is similar to the scenario in real life, where traders can freely choose the market to take part in. It also avoids the bottleneck resulting from the only one DAA.

The query load is the commodity to exchange. Exchanging is determined based on the query frequency associated with the data. The query load on a data group with k data units (records) is expressed as $L = \sum_{j=1}^{k} l_j$, where $l_1, l_2, \ldots, l_k$ are query loads for the k data units. In order to sell a query load of L_s, n data units should be first picked out on which the sum of the query load equals L_s. When query load L_s is sold from one CDN, say D_1, to another CDN, say D_2, the n data units are migrated from D_1 to D_2.

Generally, in MBA, the whole query load balancing process is as follows:

• A DTA monitors the query load upon the CDN it resides on, and sends the ask (bid) price for the load to sell (buy) to DAA.

• DAA collects ask or bid price information from DTAs distributed on various CDNs, determines transactions between traders with the market clearing algorithm (see Algorithm 1 in the following section), and reports matching results to every DTA.

• DTAs carry out data migration and update status information such as query load and storage space.

• DAA notifies the GPM (Global Partition Map) to update its partition mapping information, so that the later arrived application can get the most current location on which the required data reside. Also data index will be updated.

The above process is executed iteratively until the system arrives at a convergence, that is, a state of query load balance. It may take more than one round to converge. The time complexity of the above process is $\boldsymbol{O}(m.N^2)$, where N is the number of CDNs, and m is the number of rounds to convergence.

5 Market, bid and ask

In this section we will give the details on the MBA approach, including how to define the query load to sell, how to give a bid or ask price and how a transaction between traders is decided.

5.1 Defining the query load to sell

In a cloud database system, the basic unit for migrating is a partition. Suppose there are two partitions P_1 and P_2; their query loads are $L_{P_1} = \sum_{i=1}^{k} l_1^i$ and $L_{P_2} = \sum_{j=1}^{m} l_2^j$, where l_1^i is the query load of row R_i ($\in P_1$) and l_2^j is that of row R_j ($\in P_2$). Usually, $l_1^i \neq l_2^j$, and thus $L_{P_1} \neq L_{P_2}$. That is, different partitions cannot be identified as the same type of commodity simultaneously since the query loads associated with them are very different from each other.

However, most work in the computer-simulating-based market mechanism design [15,16], considers only one type of goods in the market and trade one single unit at a time. Thus they are not inapplicable here.

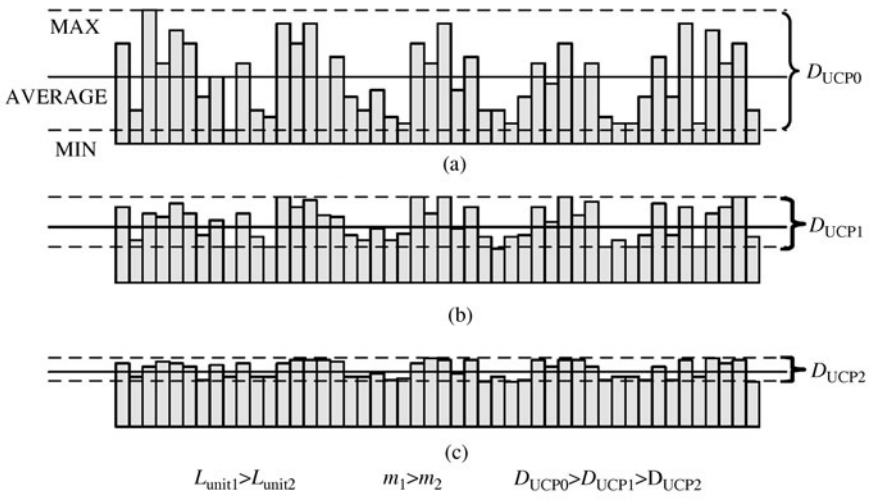

Figure 3 Query load balancing with different L_{unit}. (a) Query load is unbalanced; (b) query load balancing process stops after m_1 rounds with L_{unit1}; (c) query load balancing process stops after m_2 rounds with L_{unit2}.

To deal with the problem, we first define a standard trading amount L_{unit} used by all traders in the market, which is the amount of query load a trader has to buy or sell at a time. And then, we provide a simple yet efficient function GET_PARTITIONS_TO_SELL() to get as quickly as possible the target partitions to sell for a data node.

Partitions to sell in Γ is treated as a single unit for trading. The little difference in load value of every Γ is generally negligible, so the goods in the market can all be seen as of the same type.

Standard trading amount L_{unit} has direct effects on the speed of convergence. The smaller L_{unit} is, the better balanced state the system can achieve. The reason is as follows. Suppose the unused computing powers of various CDNs are $r_1, r_2, \ldots, r_N$, and the average value is $r_{\text{avg}} - (r_1 + r_2 + \cdots + r_N)/N$ when the system obtains load balancing. Note that the strict balanced state $r_1 = r_2 = \cdots = r_N = r_{\text{avg}}$, is hard to achieve. Suppose D_{MAX} is the CDN with the most unused computing power and D_{MIN} with the least. The following two conditions could not be true at the same time:

1) $r_{D_{\text{MAX}}} - r_{\text{avg}} \geqslant L_{\text{unit}}$ and

2) $r_{\text{avg}} - r_{D_{\text{MIN}}} \geqslant L_{\text{unit}}$,

otherwise there will be a transaction between D_{MAX} and D_{MIN}. However, as L_{unit} decreases, a value of L_{unit} can be found that enables the above two conditions to be true at the same time. Then there will be a transaction between CDNs again, and a better balanced state can be achieved.

For example, as is shown in Figure 3, unused computing powers of various CDNs are very different from each other at first. It takes m_1 and m_2 rounds to converge when the standard trading units are L_{unit1} and L_{unit2} respectively. Here $L_{\text{unit1}} > L_{\text{unit2}}$ and $m_1 < m_2$. After the process of load balancing stops, the largest difference of unused computing power between CDNs, are D_{UCP1} and D_{UCP2} respectively, and we have $D_{\text{UCP1}} > D_{\text{UCP2}}$.

5.2 Bid and ask

Data trading agent (DTA) is further classified into two types: buyer and seller. Each data node (Figure 2) is associated with these two types of agents. The seller sells its load (or sends data records), and the buyer buys load from the sellers (or receives data records). Thus a data node has two self-interested traders with diametrically opposite goals. A seller from one data node can transact with a buyer from another data node, but never with the buyer from the same data node. Executing a transaction means some records would be relocated from the data node associated with the seller to the data node associated with the buyer.

To make the above scheme work, the limit prices of the two traders for each data node should be deliberately set and updated. When the query load on a data node is high, the seller on it can make transactions with buyers from lightly-loaded data nodes, and similarly, when the query load on a data

node is low, the buyer on it can make transactions with sellers from heavily-loaded data nodes. Here the limit price, or known as private value, is the highest (lowest) price a buyer (seller) is prepared to buy (sell) a kind of goods. A buyer is constrained to purchase a price no more than its limit price, and a seller to accept a price no lower than its limit price.

For data node D_i with constant computing capacity c_i, suppose the seller on it is denoted as S_i and the buyer on it B_i, and its load is L_{D_i} and its unused computing power is $r_i = (c_i - L_{D_i})$, then the limit price of S_i is set to be $v_{S_i} = \alpha.r_i + \beta$, and the limit price of B_i is set at $v_{B_i} = \alpha.r_i - \beta$, where $\alpha \in [0, 1]$ and β is a small positive number. α is introduced to make sure that only a fraction of remaining computing power will be offered to help heavily-loaded data nodes, otherwise a lightly-loaded server may have the risk of becoming a heavily-loaded machine; and β is introduced to make sure that there is no possibility that the seller and the buyer from a same data node can make a transaction when they both make shouts in the market. Having two traders with opposite interests guarantees that it is up to the auction mechanism to determine whether the data node is competitive on the selling side (heavily-loaded) or competitive on the buying side (lightly-loaded).

In a real market, however, traders are usually not willing to offer their true value (limit price or private value) in an auction process, for the reason of earning profits. If a buyer pays more than its private value, or a seller accepts less, then the trading will be considered to be at a loss. Therefore, at first, a buyer will usually give a lower bid price than its private value and a sell will give a higher ask price than its private value. Based on this observation, the bid (ask) price a buyer (seller) is prepared to bid (ask) is a monotonic function of their private value for the good in question.

Suppose the private values of a buyer and a seller are v_{B_i} and v_{S_i} respectively. Then the bid price p_{B_i} and ask price p_{S_i} are defined as follows:

$$p_{B_i} = \Psi(v_{B_i}) \text{ and } p_{S_i} = \Psi(v_{S_i}). \tag{1}$$

A bid or ask usually takes the following form:

$$V = \{\text{ID},\ \text{commodity},\ \text{amount},\ \text{price},\ \text{storage}\}. \tag{2}$$

In our method MBA, a bid vector is $V_B = \{B_i, L_{\text{unit}}, 1, p_{B_i}, \delta_{B_i}\}$ and an ask vector is $V_S = \{S_i, L_{\text{unit}}, 1, p_{S_i}, \eta_{S_i}\}$. δ_{S_i} is the storage requirement for the data sold by S_i. η_{B_i} is storage limit for buyer B_i. If a transaction between B_i and S_j is determined, then $\eta_{B_i} > \eta_{S_j}$. For a bid, a buyer only knows the total loads (L_{unit}) it will buy, but will know nothing about which partitions constitute the query loads of L_{unit} until a transaction is decided. In contrast, a seller knows which partitions to sell at the time of ask, which are the partitions in Γ. Through bid and ask, the decentralized control for load balancing is achieved.

5.3 Market clearing

Data auctioning agent (DAA) can be seen as a market institution which lays down rules about what the traders can do and how the final allocation of commodities (i.e., data) gives the actions of the traders. DAA receives messages about the price information from traders nodes. In the case of a bid, the price information may be an offer to buy, and in the case of an ask may be an offer to sell. DAA is also responsible for market clearing; that is, DAA will decide a transaction between traders whose bid and ask values cross. In other words, if there is a transaction between seller S_i and buyer B_j, and the prices of the ask and the bid are respectively p_{S_i} and p_{B_i}, then we have $p_{B_i} > p_{S_i}$.

Suppose there are buyers $B_1,\ B_2, \ldots, B_n$, and sellers $S_1,\ S_2, \ldots, S_n$, market clearing maximizes $\sum TRANSACTION(B_i, S_j)$ subject to

$$p_{B_i} > p_{S_i} \text{ and } \eta_{B_i} > \eta_{S_j},$$

where $TRANSACTION(B_i, S_j)$ denotes a transaction between buyer B_i and seller S_i, η_{B_i} denotes the available storage space of buyer B_i, and η_{S_j} represents the required storage space of seller S_i.

The market clearing aims to maximize the total amount of transactions between buyers and sellers. $\eta_{B_i} > \eta_{S_j}$ ensures that the buyer B_i has enough storage space to store the data from the seller S_i.

Algorithm 1 shows the process of market clearing. $\mathrm{MAX}(V_{B_i}.price) - \mathrm{MIN}(V_{S_j}.price) < \theta$ (θ is a system-defined threshold of profits) is used to ensure that the market clearing begins only if there are enough trading profits. Generally, there will be few or even no trading profits when the system arrives at or near a balanced state. Under such circumstances, market clearing algorithm will not do any match between buyers and sellers. By setting different values of θ, the system will arrive at different states. With the increase of θ, the system will arrive at a state farther from the strict balance. Note that a strict balanced state requires more communication cost than a near optimal state. θ can be set according to the system requirement.

Algorithm 1: MARKET_CLEARING()

```
INPUT: the bid vectors V_{B_i} of all buyers;
           the ask vectors V_{S_i} of all sellers;
01. sort the elements in BL
        according to price in descending order;
02. sort the elements in AL
        according to price in ascending order;
03. if (MAX(V_{B_i}.price) − MIN(V_{S_j}.price) < θ)
04.     then return;
05. for each bid vector V_{B_i} in BL
06.     if (V_{B_i}.price > V_{S_j}.price
                and V_{B_i}.storage > V_{S_j}.storage)
        then
07.         output TRANSACTION(B_i, S_j);
08.         remove V_{B_i} and V_{S_i} from BL and AL;
09.         break;
```

6 Experimental study

In this part, we will report our experimental results. In our prototype system, each CDN runs a DB2 V8.0 database. As is described in Section 3, the CDMN is responsible for redirecting an incoming query request to a proper CDN to get the required data. Table 1 gives an overview of the hardware configurations of our experimental environment. The processing rate is measured by the number of queries processed by 600 threads in a second. Here the relative processing rate is computed by setting the processing rate of the slowest computer to 1. The distances between these CDNs are also available, which will be used in data migrating process. All the CDNs are assumed to be connected by a communication network with a bandwidth of 100 Mbps.

For the purpose of this study, we obtain customer data from a mobile communication company. Basic information of 6000000 users are contained in the tables, covering attributes like the basic ID of the user, the current location of the user and IMSI (International Mobile Subscriber Identification Number). The table group is partitioned and spreads across various CDNs. Queries are submitted through the application clients, and then be routed to certain CDNs. A query created by our program includes an ID number used as the key for searching the user location information in the cloud database. All queries are generated according to a Zipf distribution with the Zipf factor varying from 0.7 to 0.9. This makes the simulated scenario be more tally with practice. Because in real applications, only a few data are queried with high frequency during a period of time, a medium amount of data with middle query frequency, and a large amount of data are queried not so continually.

To simulate the queries in the real applications, we generate 600 threads to send queries concurrently to the cloud database system, with each thread carrying several SQL statements. It is found from our experiments that, if the overall queries amount to 6000, the system resources are consumed to the largest extent. Hence the total number of the generated queries is varied from 3000 to 6000, which results in the system load level changing from 0.5 to 1.0, defined as $L_{\mathrm{sys}} = (\sum_{i=1}^{N} L_i)/(\sum_{i=1}^{N} \mu_i)$, where L_i is the query

Table 1 System configuration

Computer hardware	Number	Processing capacity(relative)
CPU 2.0GHz P4, 512M RAM, Windows 2003 Server SP1	1	1
CPU 4 1.8GHz AMD Opteron 865, 18GB RAM, Windows Server 2003 X64 SP2	1	5
AMD Athlon 64 3200+, 1G RAM, Windows XP Professional 2002	50	2

load of a CDN and μ_i the processing capacity of it.

6.1 Performance comparison

In this part, we will first introduce the performance index, and then give the methods for comparison and the experimental results.

When it comes to performance evaluation on various approaches of query load balancing, query response time and balance metric are two main aspects to be concerned. In our experiments, a fairness index I proposed in [17] is used, which is $I = ([\sum_{i=1}^{n} F_i]^2)/(n\sum_{i=1}^{n} F_i^2)$, where F_i is the expected response time of node i. If the expected response time is the same for all nodes, then there is $I = 1$, which means that the system achieves fair distribution. Otherwise, if the expected response time differs significantly from one node to another, then I decreases, meaning that the system is suffering from load imbalance to some extent.

It is good to start the simplest scenario where 1) the auctioneer is stationary; that is, it does not change over time; 2) each trader knows where the auctioneer is; 3) the query frequency history of all the data is available; 4) the distribution of queries upon the data records is stationary; and 5) the physical topology of the database servers is not considered.

Based on these assumptions, we can now compare market-based approach with other methods. RR and DAH [18] are static methods, which means there will be no data reallocation process.

In RR, an incoming partition P is allocated to the 'next' available data node D in a ring, without considering whether D is currently heavily-loaded or not, or how many query loads will be imposed on D by P. For this purpose, an iterator is used which cycles through the ring of data nodes in a cloud database system. The search starts at the current position of the iterator, which is increased after a proper data node is found for the locating of P.

DAH is an abbreviation for data allocation using a hash-based heuristic technique, and is used to allocate data fairly for load balancing in the system. To achieve this, the history of query load on the data is used for load prediction. In DAH, data partitions are allocated to different data nodes by a hash-based heuristic technique.

Then we will compare MBA with a dynamic data allocation method, DDMC (dynamic data migration by centralized control) [18] in a more complex environment, where query patterns are ever-changing over time. DDMC is a dynamic data reallocation policy with a central controller proposed in our previous work, which collects the global information of all servers such as query load and processing power, and then makes decisions on data migrating by heuristic algorithm. In order to reduce cost, all the above jobs are carried out periodically instead of in real-time.

6.1.1 *Comparison with static methods*

From the experimental results, MBA achieves much better performance than other static methods, such as RR and DAH. The much difference of MBA from both RR and DAH is that, in MBA, there will be data exchanging among various CDNs during the data allocation process, but it is not the case for both RR and DAH, i.e., for them data remain in a CDN instead of being relocated from one to another. As is described above, this experiment is based on the assumption that query load distribution history is available, and the distribution of queries upon the data records is stationary, so that the method of DAH can perform best here. Note that DAH performs worse than RR in some cases if query load distribution is evolving along with time, which is also proved by our experimental results. But DAH is not the focus of this paper, and thus experimental results on such cases will not be included here.

In the experiments, the number of CDNs is fixed at 10, and the query arrives after the data allocation process stops. The system load level changes from 0.5 to 1.0. There are several aspects to evaluate:

• Average response time of cloud database system as a whole. As Figure 4 shows, the average response time of RR changes the most during the whole process of load adjustment. When L_{sys} equals 0.5, it is 0.152 s, but it reaches a high level of 0.326 s when L_{sys} increases to 0.8. MBA achieves the best average response time under all system load levels compared with RR and DAH. It is only 0.05 s for the case of $L_{\text{sys}} = 0.5$, and slowly increases to 0.104 s along with the changing of L_{sys} from 0.5 to 1.0.

• Average response time of individual CDN. Since the hardware configurations for each CDN in our experiment are all the same, the average response time of each individual CDN should not have much difference from each other, when there is a load balance among various CDNs. Here we provide the average response time of 10 CDNs, as is shown in Figure 5 and the system load level is 0.7. It can be concluded that MBA is better at query loading balancing than RR and DAH. Because the difference in the response time between the fastest CDN (the first, seventh and eighth CDN for RR, DAH and MBA respectively) and the slowest one (the fifth, fourth and fourth CDN for RR, DAH and MBA respectively) is decreased, and the load level of each CDN is moderated to average system load level as much as possible. As far as standard deviation (sd) is concerned, MBA achieves the lowest level, while RR is with the highest level, as is shown in Figure 5. Here there is also an interesting thing that the average response time of RR for the CDNs of 1, 2, 7 and 9, is even faster than that of both DAH and MBA. The reason is that these CDNs are allocated by RR with too few query loads, which results in lightly-loaded CDNs.

• Fairness index. With respect to fair index, Figure 6 shows that RR is with the lowest level of fairness index, which changes between 0.801 and 0.543 when system load level increases from 0.5 to 1.0. DAH achieves a moderate level of fairness index ranging from 0.911 to 0.778. MBA remains a high level of fairness index between 0.980 and 0.916 during the whole process, because there exist neither heavily-loaded nor lightly-loaded CDNs when the system load level does not exceed the processing capacity of the cloud database system.

6.1.2 *Comparison with dynamic method*

In this part, we will show that MBA as a decentralized-control based dynamic method, can achieve better performance than other centralized-control based dynamic method such as DDMC. Also, in order to show the performance advantage of dynamic method over static one, we will take DAH into consideration. Here the number of CDNs is fixed at 10, and the query load pattern varies continuously. The system load level changes from 0.5 to 1.0. There are two aspects to evaluate:

• Average response time of cloud database system as a whole. As Figure 7 shows, dynamic methods MBA and DDMC can achieve much better performance than static method DAH. This is due to the reason that DAH is based on the history information of query load distribution, and therefore it will be no longer as much useful when query load pattern changes along with time. In contrast, MBA and DDMC can adapt to such variation effectively. Furthermore, MBA has a better performance than DDMC, and the difference will be more with the increase in the number of CDNs (shown in the experimental results later).

• Fairness index. As Figure 8 shows, MBA achieves higher fairness index than both DAH and DDMC. As system load level increases from 0.5 to 1.0, the fairness index of DAH becomes worse in a faster speed than both MBA and DDMC. This shows the effectiveness of dynamic method in achieving load balancing.

6.2 Detailed report on MBA

In this part, we will report the experiments of dynamic data migrating with MBA, including its flexibility and scalability under different testing environments. We set the transaction rounds in a day at 20. Within each round a trader can only make one offer, either a bid for a buyer or an ask for a seller. In each transaction, there is only one goods (i.e., partition) to be exchanged. Also, for the convenience of com-

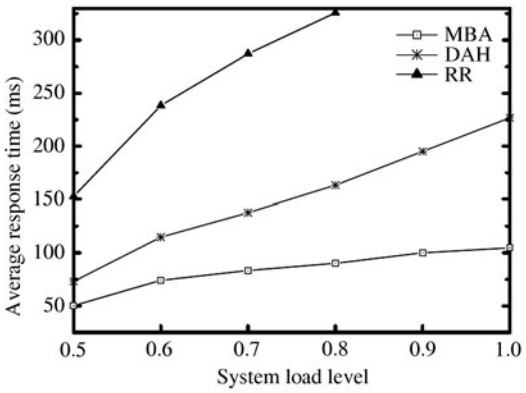

Figure 4 Average response time.

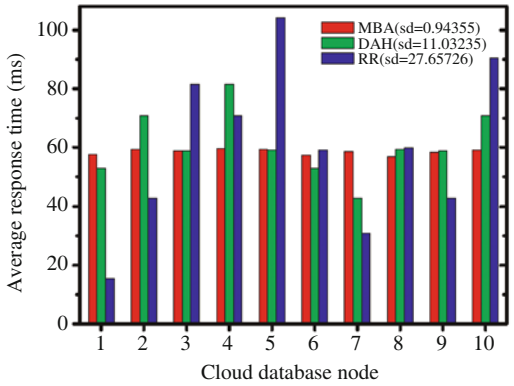

Figure 5 Average response time of individual CDN.

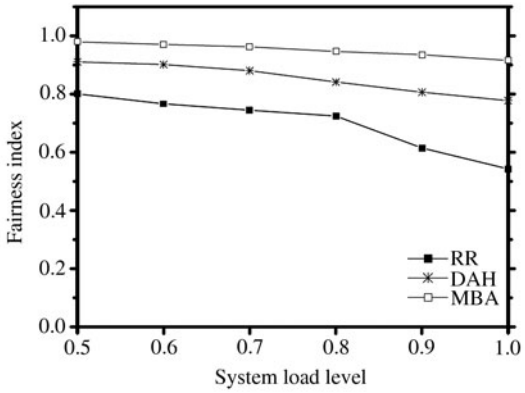

Figure 6 Fairness index.

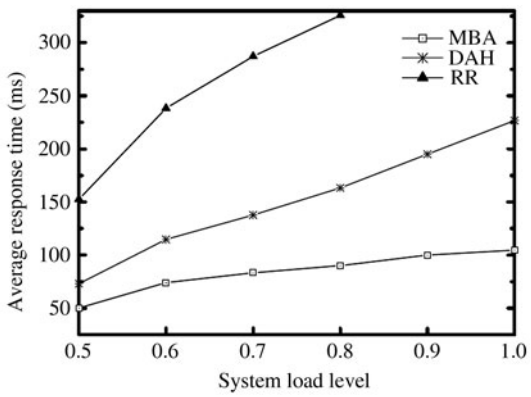

Figure 7 Average response time.

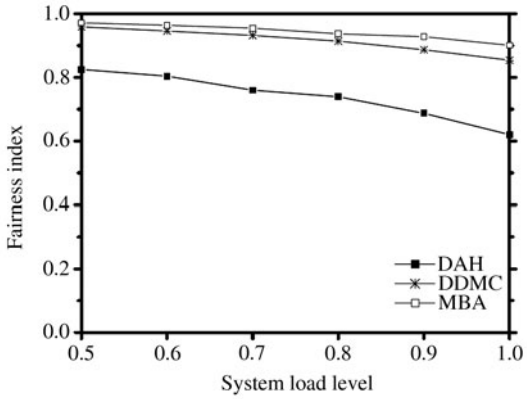

Figure 8 Fairness index.

parison, we use the relative value of query load for each CDN. In the auction, the bid and ask price for a trader is computed using Eqs. (1) and (2) given in Subsection 5.2. Since each CDN has the same hardware power, c_i is the same for each CDN. Here α and β are fixed at 0.8 and 0.05 respectively.

Figure 9 demonstrates the query load balancing processes when adopting the auction mechanism of CDA. When the system load level is 0.7, the query load levels (relative values) of CDNs are distributed between 1 and 100, and the processing capacities of CDNs range from 20 to 100. In round 0, the unused computing powers of CDNs are very different from each other, whereas they are becoming closer and closer along with time. In the 15th round, unused computing powers of various CDNs arrive at a convergence to equilibrium, which is also the load balanced state defined in our approach.

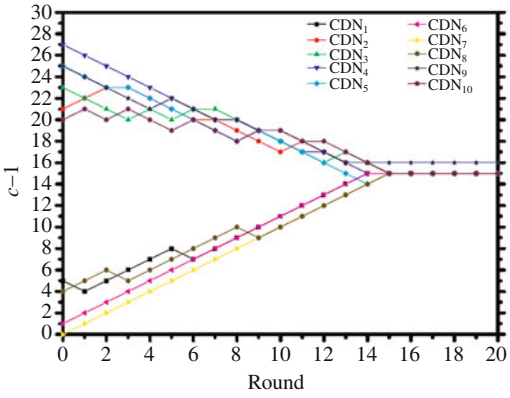

Figure 9 Load balancing for 10 CDNs.

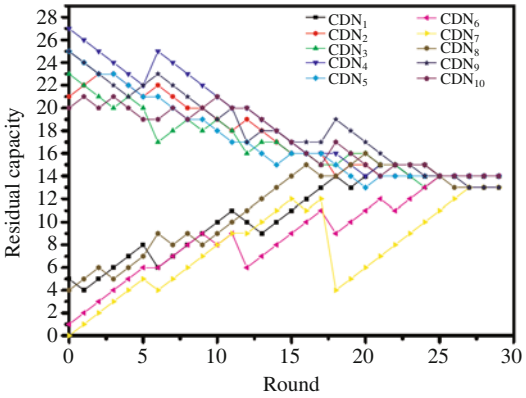

Figure 10 Dynamically changing query load distribution.

We also change the distribution of query load to see how MBA performs desirably under such variation. As shown in Figure 10, in the first round R_1, unused computing powers (relative values) of 10 CDNs are 5, 21, 23, 27, 25, 1, 0, 4, 25 and 20 respectively. Unused computing powers of various CDNs begin to come close to each other resulting from the auction mechanism. Then in the sixth round R_6, there occurs a variation on the distribution of query load, which leads to the change of unused computing powers of CDNs. Based on this new distribution, CDA works again to arrive at a new balance. Similarly, in the rounds of R_{12} and R_{18}, the same process happens again. Once such variation occurs, it means that CDA carries out its job on a new base of query load distribution. In the 29th round, this process arrives at a convergence of equilibrium; thus there will be little change in query load distribution afterwards. From Figure 10, we can see the fluctuations of the query load curves for various CDNs. No matter how query load distribution changes, in the end, the query loads of various CDNs will converge to an equilibrium. Therefore, it can be concluded that MBA can effectively deal with query load evolution.

We measure the flexibility and scalability of MBA by changing the data node topology during the load balancing process. For flexibility, it mainly considers how MBA adapts itself to the changing of CDN topology, i.e., the adding or removing of CDNs. For scalability, it concerns how MBA performs with a large increase in the number of CDNs. Scalability is especially important in the cloud database environment.

Figure 11 shows the query load balancing process when dynamically adding or deleting CDNs. Initially, there are 10 CDNs with unused computing powers (relative values) of 5, 21, 23, 27, 25, 1, 0, 4, 25 and 20 respectively. In round 6, two more CDNs, namely N_1 and N_2 with unused computing powers of 30 and 25, are added to the system. In round 17, some two CDNs are deleted from the system, and the data residing on them is reallocated to N_1 and N_3. From Figure 11, it can be seen that MBA can effectively deal with the variation of CDN amount.

It can be easily proved that the number of CDNs has no influence on the speed of the convergence, which means MBA has high scalability. In Figure 9, there are 10 CDNs, and $\lfloor(\mathrm{MAX}_{i=1}^{N}|r_i - r_{\mathrm{avg}}|)/L_{\mathrm{unit}}\rfloor$ equals 15, hence it takes 15 rounds to converge. In Figure 12, there are 50 CDNs, and $\lfloor(MAX_{i=1}^{N}|r_i - r_{\mathrm{avg}}|)/L_{\mathrm{unit}}\rfloor$ is 48; thus the system arrives at a converge after 48 rounds. For both cases, the number of CDNs has no effect on the convergence speed. Furthermore, we compare the scalability of MBA and DDMC. As Figure 13 shows, MBA can achieve better performance than DDMC with the increasing number of CDN.

7 Conclusions

In this paper, we proposed a market-based approach, called MBA, to address load balancing in cloud database via data allocation and dynamic migration. Unreasonable data distribution may lead to a load-imbalanced system, and thus slow the average response time of the system. MBA successfully deals with data distribution in cloud database from a new angle. Intelligent market rules are integrated into the

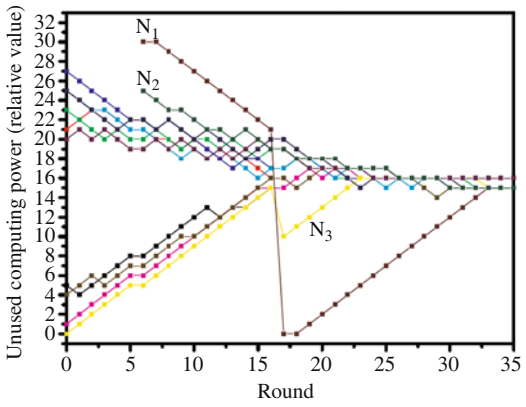

Figure 11 Dynamically adding or deleting CDNs.

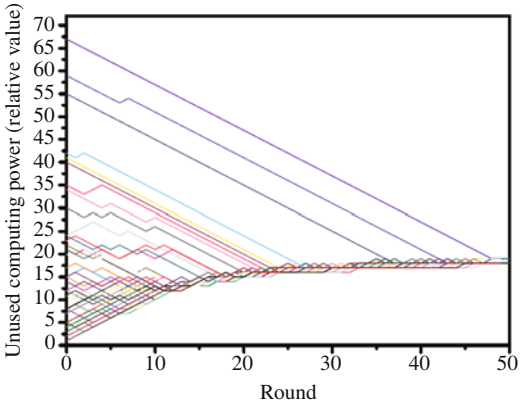

Figure 12 Query load balancing process for 50 CDNs.

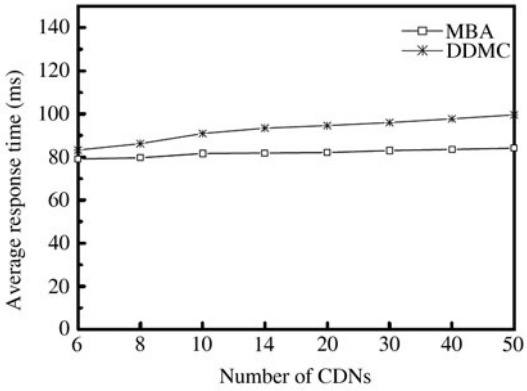

Figure 13 Average response time when varying the number of CDNs.

mechanism of data allocation and migration in the cloud database. With MBA, the system can achieve load balance efficiently and adapt to the changes over time effectively. We conducted extensive experiments on the prototype system with customer data from a mobile communication company, and the results show that MBA can achieve much better performance than other baseline methods. Furthermore, it has desired flexibility and scalability, which is especially important for the ever-changing extreme-scale cloud database environment.

Acknowledgements

This work was supported by Cultivation Fund of the Key Scientific and Technical Innovation Project, Ministry of Education of China (Grant No. 708001), Fundamental Research Funds for the Central Universities (Grant No. 2011121049), National High Technology Research and Development Program of China (Grant Nos. 2007AA01Z191, 2009AA01Z150), National Science and Technology Major Program (Grant Nos. 2010ZX01042-001-003-05, 2010ZX01042-002-002-02), National Science & Technology Pillar Program (Grant No. 2009BAH44B03), and the joint project of Microsoft SQL China R&D Center and Peking University.

References

1 IBM software strategy group. IBM Google Announcement on Internet-Scale Computing (Cloud Computing Model), 2007

2 Niu J, Cai K, Gerding E, et al. Characterizing effective auction mechanisms: insights from the 2007 tac market design competition. In: Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems, Estoril, 2008

3 Zhang Y, Kameda H, Hung S L. Comparison of dynamic and static load-balancing strategies in heterogeneous distributed systems. Comput Digit Tech, 1997, 144: 100–106
4 Lin H C, Raghavendra C S. A dynamic load-balancing policy with a central job dispatcher (LBC). IEEE Trans Softw Engineer, 1992, 18: 148–158
5 Narendran B, Rangarajan S, Yajnik S. Data distribution algorithms for load balanced fault-tolerant web access. In: Proceedings of the 16th Symposium on Reliable Distributed Systems, North Carolina, 1997. 97–106
6 Chen L C, Choi H A. Approximation algorithms for data distribution with load balancing of web servers. In: Proceedings of IEEE International Conference on Cluster Computing, California, 2001. 274–281
7 Tse S S. Approximate algorithms for document placement in distributed web servers. IEEE Trans Softw Engineer, 2005, 16: 489–496
8 Yokota H, Kanemasa Y, Miyazaki J. Fat-Btree: An update-conscious parallel directory structure. In: Proceedings of the 15th International Conference on Data Engineering, Sydney, 1999. 448–457
9 Lee M L, Kitsuregawa M, Ooi B C, et al. Towards self-tuning data placement in parallel database systems. In: Proceedings of SIGMOD Conference, Texas, 2000. 225–236
10 Feelifl H, Kitsuregawa M, Ooi B C. A fast convergence technique for online heat-balancing of Btree indexed database over shared-nothing parallel systems. In: Proceedings of the 11th International Conference on Database and Expert Systems Applications, London, 2000. 846–858
11 Watanabe A, Yokota H. Adaptive lapped declustering: A highly available data-placement method balancing access load and space utilization. In: Proceedings of the 21st International Conference on Data Engineering, Tokyo, 2005. 828–839
12 Friedman D. The double auction institution: A survey. In: The Double Auction Market: Institutions, Theories and Evidence. Boulder: Westview Press, 1993. 3–25
13 Nicolaisen J, Petrov V, Tesfatsion L. Market power and efficiency in a computational electricity market with discriminatory double-auction pricing. IEEE Trans Evol Comput, 2001, 5: 504–523
14 Tesauro G, Kephart J O. Pricing in agent economies using multi-agent Q-learning. Auton Agents Multi-Agent Syst, 2002, 5: 289–304
15 Gode D K, Sunder S. Allocative efficiency of markets with zero intelligence traders: Market as a partial substitute for individual rationality. J Political Econ, 1991, 101: 119–137
16 Cliff D, Bruten J. Less than human: Simple adaptive trading agents for CDA markets. Technical Report HP-97-155. 1997
17 Jain R. The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling. New York: Wiley-Interscience, 1991
18 Wang T J, Yang B S, Gao J, et al. Effective data distribution and reallocation strategies for fast query response in distributed query-intensive data environments. In: Proceedings of APWeb, Shenyang, 2008. 548–559