2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing

ALARM: Autonomic Load-Aware Resource Management for P2P Key-value Stores in Cloud

Can Zhang
School of Software
Shanghai Jiao Tong University
Shanghai, China
Email: zhangcan@sjtu.edu.cn

Haopeng Chen
School of Software
Shanghai Jiao Tong University
Shanghai, China
Email: chen-hp@sjtu.edu.cn

Shuotao Gao
School of Software
Shanghai Jiao Tong University
Shanghai, China
Email: shuotao.gao@gmail.com

***Abstract*—This paper presents ALARM, an autonomic load-aware resource management algorithm that can be used to manage physical machines or virtual machines in cloud, which participate in a P2P key-value store. A lot of existing key-value stores claim that they are elastic enough to scale up or down with no downtime or interruption to applications. However, the question that when the scaling up or down should take place has still not been resolved. The situation may get worse if the data store consists of hundreds of machines, for it's unrealistic for a system administrator to monitor the system and add/remove a machine manually. Fortunately, cloud computing and virtualization technology have enabled the real-time provision of virtual machines and a way of managing virtual machines without human interference. By supervising the utilization of multiple resources (CPU, memory, network IO, etc.) in virtual machines hosting the data store, our ALARM algorithm will take effect when some of the machines become overloaded or underloaded. The experiment result shows that ALARM helps the Open Chord data store, an open-source implementation of the Chord protocol, scale up and down according to the resource usage in the virtual machines.**

***Keywords*—cloud computing; P2P key-value stores; resource management;**

I. Introduction

Cloud computing is believed to change the way we get access to data, and consequently has a great impact on the database industry [1]. While the introduction of the relational model by Codd in 1970 [2] was regarded as one of the success stories of the computer science discipline, relational databases are now being criticized for their limitations in scaling up, complex data structure, and also poor query performance with SQL [3][4]. An increasing number of web 2.0 applications and the emergence of cloud computing platforms are calling for a more distributed, easily-scalable and highly-efficient data store. As a result, a growing number of developers and researchers are turning to various non-relational databases like Cassandra[5], Apache CouchDB[6], and MongoDB[7], etc. Furthermore, when Amazon published a paper on Dynamo[8], its P2P key-value storage system for internal use, key-value stores began to attract interest both in academic fields and in IT industry. Among these key-value stores, three of the best-known ones are Amazon's Dynamo, Google's Bigtable[9], and the Apache Cassandra open sourced by Facebook in 2008.

Though key-value stores are poised to be the most promising candidate databases in cloud, they have not been fully developed, and a lot of vendors and developers are working on projects to improve their fault-tolerance, enrich their data models, and so on. In this paper, we mainly focus on the resource management problem encountered in these P2P key-value stores, for example Dynamo and Cassandra. In fact, by using technologies like Consistent Hashing[10], a lot of existing key-value stores are *elastic* enough to scale up or down with no downtime or interruption to applications. However, for a typical key-value store that consists of hundreds of machines, it's unrealistic for a system administrator to monitor the system and add/remove a machine manually. Fortunately, cloud computing and the virtualization technology [11] have enabled the real-time provision of virtual machines and a way of managing virtual machines without human interference. Thus, we present ALARM - an autonomic load-aware resource management algorithm with respect to the utilization of multiple resources in the P2P data stores. On the one hand, for an overloaded virtual machine, the load-aware algorithm will add a new virtual machine to take over some load from it. On the other hand, the algorithm will try to merge a pair of underloaded virtual machines into one, and stop the other one.

This work is meaningful to both cloud platform providers and cloud platform consumers: (i) for cloud platform providers, it helps save energy in data centers by starting up or shutting down virtual machines on demand. For example, Decandia et al.[8] discussed three load-balancing strategies used in Dynamo; (ii) for cloud platform consumers, some of their applications may be data-intensive so that pulling data from cloud storage provided by cloud platform providers may be inefficient and costly. What's more, customers may be worried about the data lock-in risk if they choose to manage their data through storage APIs provided by the cloud platform. Therefore, for those who prefer to build their own P2P key-value stores in cloud, our load-balancing algorithm will help them achieve the economic efficiency [12].

A. Our Contributions

The contributions of this paper are:

1 The ALARM algorithm presented in this paper provides an autonomic way of resource management for P2P key-value

978-0-7695-4612-4/11 $26.00 © 2011 IEEE
DOI 10.1109/DASC.2011.83

stores in cloud. Traditionally, the imbalance ratio(largest load/smallest load) is a critical criterion in characterizing the load distribution in P2P data stores. As a result, several algorithms like [13][14][15] are proposed to ensure load-balancing in the storage systems. However, a skew load distribution is acceptable in the cloud environment, for we care more about the resource utilization efficiency.

2 To the best of our knowledge, this is the first resource management algorithm that considers utilization of multiple resources (CPU, memory, etc.) for key-value stores. Traditionally, load-balancing in P2P data stores only focuses on the disk usage of each machine in the system. From our point of view, disk usage is not the single important factor when measuring the load in the system. For example, memory bottleneck will surely cause a serious performance issue in the storage system.

3 The experimental result shows that our algorithm helps the data stores scale up and down without human interference. In a typical web application, the frequency of data usage varies a lot on different machines in the data store, which will result in different loads on two machines even they have the same data volume. In light of this, ALARM is based on a real-time supervision of multiple resource usage on different machines, so that it helps the system scale up and down according to the status of multiple resources.

B. Notation

In this paper, we will use the following notation.

- n Is the number of resources measured in the system
- $c_{i,p}$ Is the capacity of resource i at node p
- $u_{i,p}$ Is the utilization of resource i at node p
- opt_i Is the optimal utilization of resource i (we assume that the optimal utilization is the same for all nodes)
- λ_i Is the weight for resource i, where $\lambda_i \in [0,1]$ for all $1 \leq i \leq n$ and $\sum_{i=1}^{n} \lambda_i = 1$ holds

The rest of the paper is organized as follows. Section 2 presents the basic model of our resource management algorithm. Section 3 shows the system design of the resource management solution for P2P data stores. The experimental results of a prototype of the ALARM algorithm on top of the Chord lookup system are shown in Section 4. Section 5 surveys some related work. Finally, the paper is concluded in Section 6.

II. Basic Model

Fig. 1 shows the basic resource-management architecture for P2P key-value data stores. As is shown in Fig. 1, there are two types of nodes in our architecture, namely *virtual node* and *physical node*. Typically, data stored in a P2P data storage are in a distributed manner, and the computer which is responsible for storing part of the data is known as a node. However, in our architecture, these nodes participating in the P2P data storage are defined as *virtual node*s, and several virtual nodes can

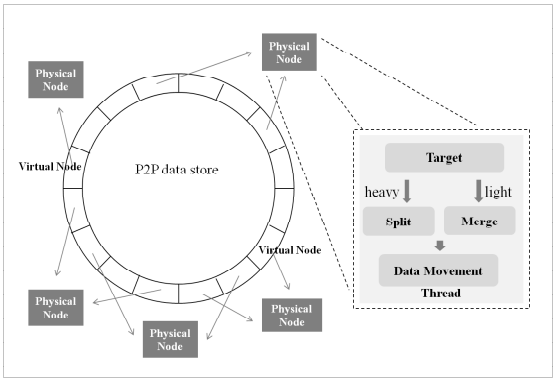


Fig. 1. Basic Resource Management Architecture for P2P data stores

be hosted on a *physical node* (e.g., a computer or a virtual machine in cloud).

Generally speaking, our resource management algorithm is designed to achieve a better utilization of multi-resources on physical nodes by transferring or splitting virtual nodes. The resource-management algorithm is composed of four operations: *target*, *merge*, *split* and *data movement*. The procedure works as follows: (i) *target*: our algorithm checks periodically whether a physical node should be targeted. A physical node will become targeted because it's either underloaded or overloaded; (ii) *merge/split*: for an targeted underloaded physical node, the resource management algorithm will try to merge another underloaded node with it; for an targeted overloaded node, part of the data will be moved to a newly started physical node; and (iii) *data movement*: finally, a data movement operation will be performed. In the following paragraphs, we will explain the four operations in details.

A. Target Operation

As mentioned above, a physical node will be targeted if it becomes underloaded or overloaded. For a physical node p, the target function $T(p)$ is defined as:

$$T(p) = \begin{cases} overloaded, & \exists i, u_{i,p} > opt_i \quad (1a) \\ underloaded, & \sum_{i=1}^{n} \lambda_i (opt_i - u_{i,p}) > t \quad (1b) \\ normal, & \text{otherwise} \quad (1c) \end{cases}$$

where t is the threshold value of the target operation, and $t \in [0,1]$. The parameters t, $opt_i (i = 1..n)$, and $\lambda_i (i = 1..n)$ are all configurable values, which can be set and tuned by the system administrator. By considering resources like CPU, network bandwidth, and memory, the decision is made according to the utilization status of these resources. If any of the resource utilization exceeds the optimal value, the physical node becomes an overloaded node(1a). It makes sense because any resource may cause a significant slowndown in the system performance. If the calculated value $(\sum_{i=1}^{n} \lambda_i (opt_i - u_{i,p}))$ exceeds the configured threshold value t, the physical node becomes an underloaded one(1b). In fact, the selection of parameters t and opt_i is non-trivial work. A larger opt_i or

smaller t value may lead to higher resource usage and thus fewer physical nodes, but may make the system vulnerable to a sudden burst of user requests. For now, we simply specify a moderate value (e.g., 0.8) to each $opt_i (i = 1..n)$ parameter, and a more rational method for parameter selection may be discussed in the future. In addition, as the target operation is executed periodically (e.g., every 30 minutes), $u_{i,p}$ represents the general resource status during the execution interval. For now, we use the average usage of each resource during the execution interval as the value $u_{i,p} (i = 1..n)$. By averaging the utilization value, it prevents the algorithm from overreacting to abnormal but benign transient situations (e.g., 95% usage of CPU within 10 seconds).

B. Merge Operation

When a physical node becomes underloaded, the merge operation will be executed. It's possible that the resources of the two nodes may have different capacities. Let $u'_{i,p1}$ and $u'_{i,p2}$ be the expected usage if the two nodes are merged to node $p1$ and node $p2$ respectively. We get the following equations:

$$u'_{i,p1} = \frac{u_{i,p1}c_{i,p1} + u_{i,p2}c_{i,p2}}{c_{i,p1}} \tag{2a}$$

$$u'_{i,p2} = \frac{u_{i,p1}c_{i,p1} + u_{i,p2}c_{i,p2}}{c_{i,p2}} \tag{2b}$$

Nodes $p1$ and $p2$ can become a *merging pair* when $u'_{i,p1}$ or $u'_{i,p2}$ is below the optimal value. Thus, the *merging pair* must satisfy at least one of the following two conditions:

$$u'_{i,p1} \leq opt_i, \forall i \in [1, n] \tag{3a}$$

$$u'_{i,p2} \leq opt_i, \forall i \in [1, n] \tag{3b}$$

If one of the conditions is met, suppose it's (3a), then we get the matching evaluator $M(p1, p2)$ as:

$$M(p1, p2) = \sum_{i=1}^{n} \lambda_i (opt_i - u'_{i,p1}) \tag{4}$$

If both of the conditions are met, the $M(p1, p2)$ is:

$$M(p1, p2) = min \left\{ \sum_{i=1}^{n} \lambda_i (opt_i - u'_{i,p1}), \sum_{i=1}^{n} \lambda_i (opt_i - u'_{i,p2}) \right\} \tag{5}$$

The smaller the matching evaluator is, the closer the resource utilization is to the optimal value. If more than one pair is found for the targeted node, the pair with the smallest $M(p1, p2)$ value will be chosen as the merging pair. If no pairs are found, it means the load on other nodes are not that light as this node, so nothing should be done.

In fact, a merging mate for the underloaded node can be found in a centralized way or a distributed way. On the one hand, a centralized controller may maintain the information of all the available physical nodes in the data storage. Thus, by querying the centralized controller, an underloaded node can get a list of available nodes and check if they meet the condition(2). On the other hand, the task of pairing two underloaded nodes can be performed by self-organizing agents: a merging agent may be generated on the underloaded node, and then finds a suitable merge mate for this node by hooping over the P2P network overlay. The second solution is more fault-proof, for the simple reason that it does not have any single point of failure. However, it's not that efficient as the first solution. Currently, we design our ALARM algorithm based on a centralized controller, and a distributed solution may be implemented in the future.

C. Split Operation

The split operation is invoked if a physical node is targeted as an overloaded node. At this time, a new physical node has to be started, for merging another node with this one will lead to an even more overloaded node. However, the split operations for a physical with one virtual node and the one with multiple virtual nodes are a little different. For a physical node which has exactly one virtual node, a new physical node will be started with a virtual node taking half volume of the data from the targeted physical node. For a physical node which hosts more than one virtual node, randomly choose a virtual node and move it to the newly added physical node.

D. Data Movement Operation

The last sub-step of both split operation and merge operation is data movement. The data movement operation could be accomplished in two ways: virtual node splitting and virtual node transferring. When splitting an overloaded physical node: if the physical node has only one virtual node, this virtual node needs splitting; otherwise, the physical node has multiple virtual nodes and one of these virtual nodes needs transferring. When merging two underloaded physical nodes, all the virtual nodes in one physical node needs transferring to the other physical node. These two ways of data movements could be achieved with the help of the virtual node joining and leaving mechanisms provided by the P2P data stores. The virtual node splitting can be implemented by adding a new virtual node which will join in and take over part of data on the overloaded physical node. The virtual node transferring can be implemented by kicking out a virtual node on one physical node and adding an identical virtual node on the other physical node. For example, if the P2P key-value store uses distributed hash table protocols as its data distribution protocol such as Chord, we can add a virtual node B whose id is between the virtual node A on the targeted physical node and A's predecessor, then the data in A is split and B takes over part of the data.

III. System Design

The resource management architecture is composed of four components: the Storage Control module, the Status Collector module, the Virtual Node, and the core ALARM module (Fig.2). First, the Storage Control module is responsible for maintaining a list of available physical nodes in the data store, and interacting with the Cloud platform to start/stop a

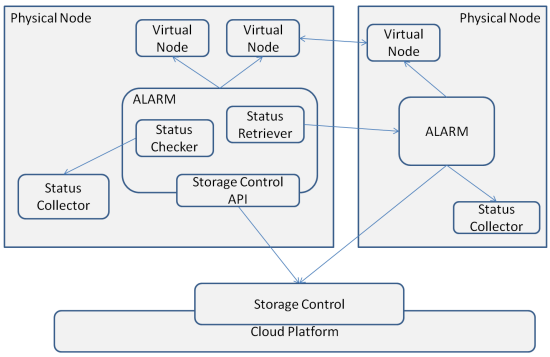


Fig. 2. ALARM System Architecture

physical node(virtual machine). Second, the Status Collector module collects the resource utilization on the local machine periodically (e.g. 20 seconds). Also, it stores the history data in a local lightweight database. Third, the Virtual Node module is responsible for data storage. The entire data in the data store are partitioned into fragments, each of which is hosted by one Virtual Node. Data operations such as insert/retrieve/update/delete are handled by the virtual node. Forth, the ALARM module is the controller in a physical node. It is responsible for handling resource adjustment, invoking operations like *target*, *split*, *merge*. In addition, it maintains information about the virtual nodes within the physical nodes. In the following paragraphs, we will explain how these modules interact with each other to complete the the resource adjustment procedure.

A. Target Operation

The Status Checker in the ALARM module checks the status collected by the Status Collector periodically, to determine whether the physical node is overloaded or underloaded. Only in these two cases, would the physical node be targeted and a new resource adjustment task be triggered: the split operation is started for the overloaded node, and the merge operation for the underloaded one. If the load on the physical node is normal (neither overloaded nor underloaded), the ALARM module will do nothing until next round of status checking.

B. Split Operation

In the split operation, the ALARM module first asks the Storage Control to start a new physical node via the Storage Control API. On the new physical node, a new virtual node will be started to take over some data from the overloaded physical node. Again, the data is transferred to the new node through joining protocol enabled by the P2P data stores.

C. Merge Operation

In the merge operation, the ALARM module first asks the Storage Control for a list of available members in the data store. Second, the Status Retriever in the ALARM module will try to get the status of each available member by communicating with the member's ALARM module. Third, the ALARM module calculates the matching evaluator(4) of each possible pair. Finally, the one with the smallest matching evaluator is selected as the merging pair, and the data movement operation will be started.

D. Data Movement Operation

In the case of overload, data are moved from the overloaded one to the newly-started one. ALARM picks only one of virtual nodes in the overloaded physical node and moves it onto the other physical node.

In the case of underload, all the virtual nodes in one physical node will be moved to the other physical node. After all the virtual nodes are moved out from one physical node, ALARM will request the Storage Control to stop this physical node via the Storage Control API.

Currently, in any period, only one adjustment task is allowed in the current system. The reason is that parallel adjustments would not only increase the complexity of the algorithm and the implementation, but also leads to an unstable system because of frequent up and down of physical nodes. In addition, the data consistency and concurrency problems are out of our concerns. On the one hand, the data movement is realized by joining/leaving of virtual nodes. Thus, it's the responsibility of the data store to maintain the data consistency among the system. On the other hand, data in the key-value stores are seldom modified, so that the data movement may not cause a serious data consistency problem in the system.

IV. Experimental Evaluation

In this section, we present the results from our experiment of the resource management algorithm on Chord, a distributed hash table protocol. Our implementation is based on Open Chord[16], an open-source implementation of the Chord protocol. In the following paragraphs, we will show our experiment environment setup, the experiment senario, and also analysis and explanations of the experiment results.

A. Experiment environment setup

A J2EE web application(Storage Control) and five physical servers with hypervisor Xen 3.4.0 are used to simulate a cloud infrastructure. Storage Control runs in Tomcat7 and provides an interface for admin operations, such as create/start/stop of virtual machines which are hosted by the five physical servers. The system configurations of physical servers are shown in Table I. Virtual machine templates of different configurations are shown in Table II. Storage Control and all the physical servers with Xen hypervisor provide a mimic Cloud Platform. Besides, Storage Control maintains the member list of the virtual machines (physical nodes) for the Open Chord data store. A corresponding client API for the features of Storage Control is created and serves as the Storage Control API. Until now, the cloud infrastructure is ready for the experiment.

Also, the Status Collector module on each virtual machine is responsible for collecting the status of the machine. The Status Collector collects utilization of CPU, Memory, Network and so on periodically(e.g., every ten seconds) and saves them to

TABLE I
SYSTEM CONFIGURATIONS OF PHYSICAL MACHINES

Physical Server	CPU	Memory	Virtual Machine Number	Xen Version
server1	AMD Athlon(tm) 64 X2 Dual Core Processor 5200+	2046M	1	3.4.0
server2	AMD Athlon(tm) 64 X2 Dual Core Processor 5200+	2046M	2	3.4.0
server3	AMD Athlon(tm) 64 X2 Dual Core Processor 5200+	2046M	2	3.4.0
server4	Intel(R) Core(TM)2 Quad CPU Q9650 @ 3.00GHz	8027M	8	3.4.0
server5	Intel(R) Core(TM)2 Quad CPU Q9650 @ 3.00GHz	8027M	9	3.4.0

TABLE II
CONFIGURATIONS OF VIRTUAL MACHINE TEMPLATES

Virtual Machine Template	Virtual CPU Number	Memory	Network Bandwidth	OS Kernel
Openchord	1	256M	1MB/s	Linux 2.6.18
Reciever	1	512M	5MB/s	Linux 2.6.18
Client	1	512M	5MB/s	Linux 2.6.18

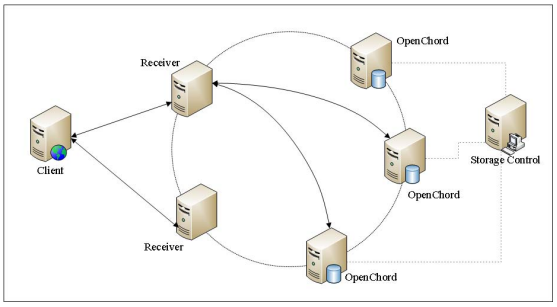


Fig. 3. Experiment senario

disk. Here, we use SIGAR[17], one open-source Java API to access system information.

B. Experiment Senario

Fig. 3 shows the basic architecture of the experiment senario. In each experiment, seven virtual machines, created from template Openchord, serve as physical nodes, which host virtual nodes in the Open Chord data store. And three virtual machines, created from template Reciever, serve as request reciever servers. Another ten virtual machines, created from template Client, are used as clients to send requests, simulating the user queries according to the 1998 World Cup Web site Access Logs [18]. We treat every url on the website as an data entry in the data store. With Chord protocol, these urls are distributed on different virtual nodes. And then, the clients' requests are sent to the request recievers and dispatched to the appropriate virtual nodes in the Open Chord data store. In addition, the 1998 World Cup Web site Access Logs record the user requests to the web site from May 1th to July 26th, and we choose the logs on June 28th as the experimental data. The number of client requests per second is shown in Fig.4.

C. Experimental Results

Our experiments show the following results:

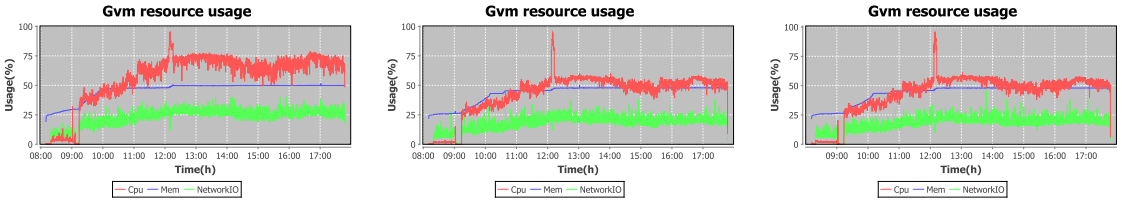
1) In a typical web application like the 1998 World Cup Web site, the frequency of data access vary a lot on different nodes in the data store(Fig.5, 6, 7).
2) Our algorithm helps the P2P key-value store scale up when the load on some virtual machines is heavy, and scale down when the load becomes light on the virtual machines(Fig.8).

There are totally 76859 url entries in the 1998 World Cup Web site. However, as is shown in Fig.5, during the interval from 13:30 to 22:00 on June 28th, only about 2000 entries were accessed by the clients. This is accordant with the resource utilization on different virtual machines shown in Fig.6 and Fig.7: Fig.6 shows the utilization of CPU, memory, and network IO of the request receivers. The load is similar to the load of client requests in Fig.5. However, in Fig.7, we find that the load coming to different physical nodes are quite different. Because the Open Chord uses consistent hashing as its data distribution algorithm, we assume that the data volume distributed to each node is almost the same. As a result, the different resource consumptions on different virtual machines demonstrate that even two nodes with the same volume of data may have totally different resource utilization.

Fig.8 shows the number of physical nodes (virtual machines) in the Open Chord with our ALARM resource management algorithm. The result shows that during the first two and a half hours, because of a relatively light load in the system, the number of virtual machines declined to only 2. However, as the peak of client requests came at about 5:30, one node was added and the number of physical nodes increased to 4. But it dropped back to 3 after the peak passed. The number of physical nodes had not changed until another peak of requests came at about 8:30. As a result, we come to the conclusion that the scale of the system is accordant with the load of the system, which demonstrates the effectiveness of our ALARM algorithm.

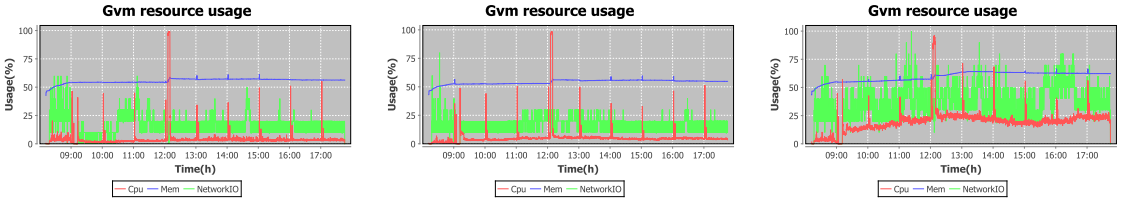
V. Related Work

Rao et al.[19] presented three schemes to do the balancing in highly heterogeneous P2P systems, namely the one-to-one scheme, the one-to-many scheme and the many-to-many scheme. Also, data are exchanged between heavy nodes and light nodes by transferring virtual servers. However, they assume that only one bottleneck resource exists in P2P systems, which is not realistic. In this paper, our algorithm supervises

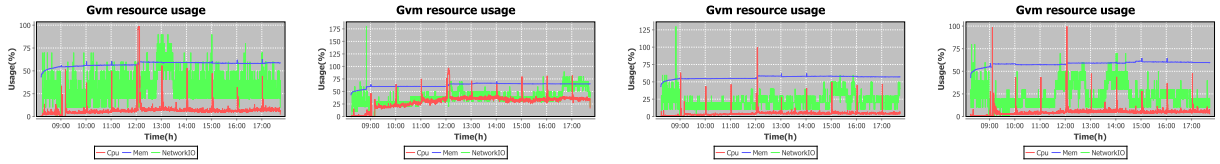


(a) vm50 (b) vm51 (c) vm56

Fig. 6. Resource utilization of Request Receivers



(a) vm57 (b) vm52 (c) vm53



(d) vm54 (e) vm55 (f) vm44 (g) vm43

Fig. 7. Resource utilization of Open Chord servers without ALARM

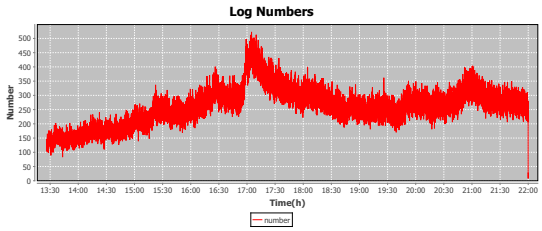


Fig. 4. Number of requests

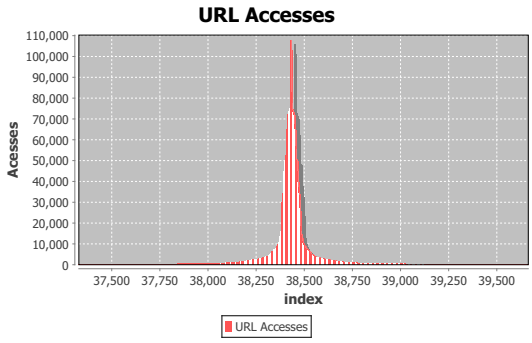


Fig. 5. URL accesses

multiple resource utilization in the system, so that we can detect utilization bottlenecks of multiple resources.

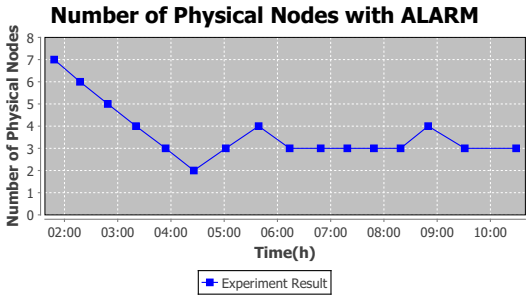


Fig. 8. Number of physical nodes

Byers et al.[13] applied the ”power of two choices”[20] paradigm to solving the DHTs' load balancing problem. Specifically, two or more hash functions are used to pick candidate peers for each item to be inserted. Among these candidate peers, the one with the least load stores the item, while the others maintain a redirection pointer to this peer. Also, the redirection mechanism is used to support other load balancing strategies like load-stealing or load-shedding. In fact, by introducing redirection pointers in the systems, the approach may end up maintaining a unpredictable number of pointers, which may cause a lose in searching efficiency and also take up too much storage space.

Karger and Ruhl[14] proposed two distributed load-

balancing protocols, balancing the distribution of the key address space to nodes and the distribution of items among the nodes respectively. The first protocol achieves a $O(1/n)$ fraction of the address space for every node with a logarithmic factor decrease of space and bandwidth usage. Also, the second protocol solves the load-balancing problem by moving data between two randomly-chosen lightly- and heavily-loaded nodes. Our work is more related to the second data-moving protocol. However, we do not always do the load-balancing work to get a small imbalance ratio. Rather, we do the resource adjustment task only if necessary, for example, when the resource usage exceeds our expectation.

Zhu and Hu[21] proposed a load-balancing scheme by using the concept of virtual servers, but in a proximity-aware manner. With the guide of the proximity information, the virtual servers are reassigned and transferred between physically close heavily loaded nodes and lightly loaded nodes. However, their scheme relies on an extra overlay on top of the DHT, which may make the P2P systems more complicated and vulnerable to failures. Again, in their work, they only care about single resource bottleneck in P2P systems.

Ganesan et al.[15] presented a Threshold Algorithm to balance range-partitioned data in parallel databases, as well as in P2P systems.Their analysis and simulation results showed that the Threshold Algorithm could achieve a desirable imbalance ratio with a relatively small amortized cost of load balancing.

Forestiero et al.[22] presented Self-Chord, a self-organizing structured P2P system. It achieves the balancing of storage responsibilities through the activity of ant-inspired mobile agents. However, the system takes a significant period of time to arrive at the steady state, if a large number of data are inserted to the system concurrently. On this occasion, the system may not work correctly.

VI. Conclusions and Future Work

In this paper, we present ALARM - an autonomic load-aware resource management algorithm with respect to the utilization of multiple resources in the P2P data stores. Considering utilization of multiple resources (CPU, memory, etc.) for key-value stores, the algorithm detects bottlenecks in the supervised resources. As it is based on a real-time supervision of the resource usage on physical nodes, our algorithm invokes a resource adjustment task when some physical nodes become overloaded or underloaded. However, a lot of work has to be done in the future. Some of them are: a decentralized way of the ALARM architecture, analysis on the behavior of the algorithm when it's applied to a large scale data store composed of hundreds of thousands of physical nodes.

VII. Acknowledgments

We thank the anonymous reviewers for their insightful feedback, the SJTU REINS research group and Qing Zheng for useful discussions and paper proof-reading.

References

[1] D. J. Abadi, "Data management in the cloud: Limitations and opportunities," *IEEE Data(base) Engineering Bulletin*, vol. 32, pp. 3–12, 2009.
[2] E. F. Codd, "A relational model of data for large shared data banks," *Communications of The ACM*, vol. 13, pp. 377–387, 1970.
[3] N. Leavitt, "Will nosql databases live up to their promise?" *Computer*, vol. 43, no. 2, pp. 12 –14, feb. 2010.
[4] M. Stonebraker, "Sql databases v. nosql databases," *Commun. ACM*, vol. 53, pp. 10–11, April 2010. [Online]. Available: http://doi.acm.org/10.1145/1721654.1721659
[5] "Cassandra." [Online]. Available: http://cassandra.apache.org/
[6] "Apache couchdb." [Online]. Available: http://couchdb.apache.org/
[7] "Mongodb." [Online]. Available: http://www.mongodb.org/
[8] G. Decandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," in *Symposium on Operating Systems Principles*, 2007, pp. 205–220.
[9] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems*, vol. 26, pp. 1–26, 2008.
[10] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, ser. STOC '97. New York, NY, USA: ACM, 1997, pp. 654–663. [Online]. Available: http://doi.acm.org/10.1145/258533.258660
[11] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*, ser. SOSP '03. New York, NY, USA: ACM, 2003, pp. 164–177. [Online]. Available: http://doi.acm.org/10.1145/945445.945462
[12] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Communications of the Acm*, vol. 53, no. 4, pp. 50–58, Apr. 2010.
[13] J. Byers, J. Considine, and M. Mitzenmacher, "Simple load balancing for distributed hash tables," pp. 80–87, 2003.
[14] D. R. Karger and M. Ruhl, "Simple efficient load balancing algorithms for peer-to-peer systems," in *ACM Symposium on Parallel Algorithms and Architectures*, 2004, pp. 36–43.
[15] P. Ganesan, M. Bawa, and H. Garcia-molina, "Online balancing of range-partitioned data with applications to peer-to-peer systems," in *Very Large Data Bases*, 2004, pp. 444–455.
[16] Distributed and M. S. G. of Bamberg University, "Open chord." [Online]. Available: http://open-chord.sourceforge.net/
[17] "Sigar." [Online]. Available: http://www.hyperic.com/products/sigar
[18] M. Arlitt and T. Jin, "1998 world cup web site access logs," 1998. [Online]. Available: http://www.acm.org/sigcomm/ITA/
[19] A. Rao, K. Lakshminarayanan, S. Surana, R. M. Karp, and I. Stoica, "Load balancing in structured p2p systems," in *Peer-to-Peer Systems*, 2003, pp. 68–79.
[20] A. W. Richa, M. Mitzenmacher, and R. Sitaraman, "The power of two random choices: A survey of techniques and results," *COMBINATORIAL OPTIMIZATION*, vol. 9, pp. 255–304, 2001.
[21] Y. Zhu and Y. Hu, "Efficient, proximity-aware load balancing for dht-based p2p systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, pp. 349–361, 2005.
[22] A. Forestiero, C. Mastroianni, and M. Meo, "Self-chord: A bio-inspired algorithm for structured p2p systems," in *Proc. 9th IEEE/ACM Int. Symp. Cluster Computing and the Grid CCGRID '09*, 2009, pp. 44–51.