A DHT-based Scalable and Fault-tolerant Cloud Information Service

Radko Zhelev
Institute on Parallel Processing
Bulgarian Academy of Sciences
Sofia, Bulgaria
zhelev@acad.bg

Vasil Georgiev
Faculty on Mathematics and Informatics
University of Sofia "St. Cl. Ochridsky"
Sofia, Bulgaria
v.georgiev@fmi.uni-sofia.bg

Abstract — **In this paper, we present an Information Service designed for maintaining resource information in Cloud datacenters. We employ a P2P cluster of super-peers that share the resource information and load of related activities in a Distributed Hash Table (DHT) based manner. Our DHT does not employ the standard keyspace range-partitioning, but implements more complicated algorithm to enable better distribution and fault-tolerance in the Cloud datacenter context. We implement a prototype of the proposed system and conduct comparative measurements that illustrate its scalable and fault-tolerant capabilities.**

Keywords – Cloud, Information Service, DHT, Scalability

I. Introduction

A Grid Information Service is software component, whether singular or distributed, that maintains information about resources in a distributed computing environment [1]. An Information Service has an Update Interface for populating resource data by Producers of resource information and a Query Interface for retrieving it by interested Consumers - system administrators, resource reservation and capacity planning tools, job schedulers, etc.

In this paper, we present an Information Service that is suitable for maintaining data about resources in a Cloud datacenter. It overcomes many limitations of existing Grids solutions taking advantage from the Cloud-specific context. Our system is formed of a P2P cluster of dedicated super-peers [2], where datacenter resource information is structured as a Distributed Hash Table (DHT). Our DHT employs a non-traditional keyspace partitioning algorithm that trades off better performance and fault-tolerance capabilities for disadvantages that are not of importance to the Cloud.

The remainder of the paper is organized as follows: related work on Grid Information Services and peer-to-peer based resource discovery is provided in Section 2. Section 3 presents our system architecture, distribution algorithm and relative use-cases. In Section 4, we describe the prototype implementation and technology. Results from experimental evaluation are presented in Section 5. Discussion on our approach and outlook to future research complete the paper.

II. Related Work Overview

A. *Information Services Organization*

A taxonomy based on system organization [3] classifies the Information Services into – centralized, hierarchical and decentralized. Centralization refers to the allocation of all query processing capabilities to single resource. All lookup and update queries are sent to a single entity in the system. Systems including R-GMA [4], Hawkeye [5], GMD [6], MDS-1 [7] are based on centralized organization [3]. Centralized models are easy to manage but they have well known problems like scalability bottleneck and single point of failure. Hierarchical organizations overcome some of these limitations at the cost of overall system manageability, which now depends on different site specific administrators. Further, the root node in the system may present a single point failure similar to the centralized model. Systems including MDS-3 [8] and Ganglia [9] are based on hierarchical organization. Performance evaluation of most popular Grid solutions – R-GMA, MDS-3 and Hawkeye, could be found at [13]. Decentralized systems, including P2P, are coined as highly scalable and resilient, but manageability is a complex task since it incurs a lot of network traffic. Two sub-categories are proposed in P2P literature [10]: unstructured and structured. Unstructured systems do not put any constraints on placement of data items on peers and how peers maintain their network connections. Resource lookup queries are flooded (broadcasted) to the directly connected peers, which in turn flood their neighboring peers. Queries have a TTL (Time to Live) field associated with maximum number of hops, and suffer from non-deterministic result, high network communication overload and non-scalability [17]. Structured systems like DHTs offer deterministic query search results within logarithmic bounds on network message complexity.

B. *Distribute Hash Tables (DHT)*

The foundations of DHT are an abstract keyspace and a partitioning scheme that splits ownership of this keyspace among the participating nodes [11], see Figure 1. Indexing could be one-dimensional or multi-dimensional, i.e., based on preliminary defined set of multiple search attributes [18]. Each node maintains a set of links to other nodes (neighbors), thus forming an overlay network. A lookup query is redirected to the neighbor that is owner of closest keyspace to the searched key, until the responsible node for that key answers the query. The keyspace partitioning has the essential property that removal or addition of one node changes only the set of keys owned by nodes with adjacent portions, and leaves all other nodes unaffected. Since any change in ownership typically corresponds to bandwidth-intensive movement of objects stored in the DHT from one node to another, minimizing such reorganization is required

This research is supported by the National Science Fund Proj. ДДВУ02/22-2010

Copyright (c) IARIA, 2011.

to efficiently support fault-tolerance and high rates of churn (node arrival and failure) [12].

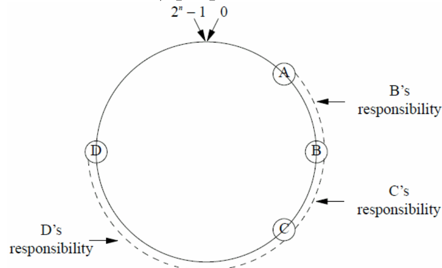

Figure 1. DHT keyspace partitioning ring.

III. SYSTEM OVERVIEW

A. System architecture

Our system architecture has three layers - Figure 2. In the bottom layer are the Producers of resource information. These are the datacenter nodes with all hardware and software resources as well as any physical or logical entities that produce resource information updates. Producers use the Update Interface to provide resource status updates to the System on the upper layer. The Information Service in the middle layer is formed by a cluster of dedicated nodes (super-peers), each of which having a local storage. Resource data is distributed within the cluster and stored by the cluster nodes in a deterministic way. The up-most layer is consisted of Consumers of resource information that use the Query Interface to retrieve data from the Information Service.

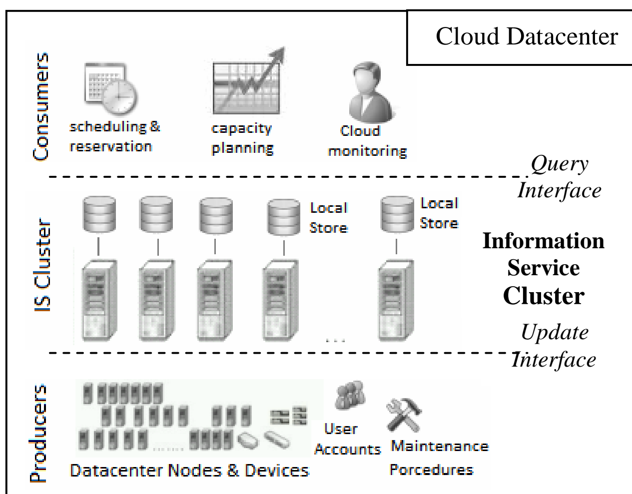

Figure 2. System architecure overview.

B. The Information Service Cluster

Our Information Service is a distributed system formed of a cluster of dedicated nodes. Every node maintains connections to all others (scheme *every-to-every*). The size of the cluster, in terms of number of participating nodes, may vary in accordance with the volume of the datacenter and the amount of resources subject of monitoring. All nodes exchange periodic pings with each other to detect if some participant gets down or becomes inoperable. Thus, we designate two states of a cluster node – *available* (1) and *not available* (0). We define a term **cluster state** as follows: considering we have a cluster of N nodes, the system orders them in a sorted sequence, assigning static index to each node $\{C_0, C_1, ..., C_N\}$. A cluster state is represented by a list of N Booleans showing the availability of each node at the respective index. We can consider that every node in the cluster 'knows' the entire cluster state, since everyone can detect if any other node goes down - when response to the ping is not received in a predefined time frame.

$$\text{Cluster State} = \{b_0, b_1, \dots b_{N-1}\},\ b_i = 0 \text{ or } 1. \quad (1)$$

C. DHT Keyspace partitioning algorithm

Our cluster acts as a DHT, i.e., every node is responsible for a certain subpart of the whole set of resources within the datacenter. Respectively, every resource has a certain responsible node where its data is stored. Our DHT employs a non-traditional keyspace assigning algorithm. It is one-dimensional and based on the resource id as follows.

We build all permutations of cluster node indexes and order them lexicographically [15], assigning to each permutation an index number. Thus, for N nodes we have N! in count permutations in the following lexicographical order:

$$P_0 = C_0, C_1, \dots, C_{N-2}, C_{N-1}. \quad (2)$$
$$P_1 = C_0, C_1, \dots, C_{n-1}, C_{n-2}.$$
$$\dots$$
$$P_{n!-1} = C_{N-1}, C_{N-2}, \dots, C_1, C_0.$$

We then use some well-defined hash function that calculates an integer number out from the resource id.

$$\text{IDHash} = \text{hashFunction}(\text{ResourceId}). \quad (3)$$

Any hash function [16] that produces chaotically spread integers would do the job. The remainder produced by dividing the hash code to the number of permutations (N!) would give us a certain permutation index from the lexicographical order of permutations:

$$r = \text{IDHash mod } N!,\ r \in [0, N!-1]. \quad (4)$$

As a result, we have a distribution function (d) that maps resource ids to certain permutations of cluster nodes:

$$d(\text{ResourceId}) \text{ -> } P_r = \{C_{r0}, C_{r1}, \dots C_{rN-1}\}. \quad (5)$$

The mapping of resource id to a certain permutation of nodes (Formula 5), we interpret as follows: C_{r0} is the primary responsible node for that resource and must handle it. C_{r1} is considered secondary responsible (fault-tolerance) node and overtakes the handle upon the resource when C_{r0} is not operable. C_{r2} is third responsible, ready to handle the resource when C_{r0} and C_{r1} are not available, and so forth.

Copyright (c) IARIA, 2011. ISBN: 978-1-61208-171-7

It could be analytically proven that as far as produced hashes are chaotically spread integers, this algorithm leads to a normal distribution of resources over the responsible nodes for any state of cluster. We have chosen to keep analytical discussion beyond the scope of this paper. It could be also proven, if not considered obvious, that our algorithm preserves the essential property of DHTs that appearing and disappearing of a cluster node concerns only the set of keys owned by this very node. This means rebalancing would be done with minimal number of redirections and without redundant swapping of responsibilities.

D. *Fault-tolerance*

In previous section, we defined how Fault-tolerance responsibilities are rebalanced, but data can not actually survive node failures if not being replicated. To enable Fault-tolerance, we define that system could be configured with predefined 'level of replication' (LR) denoting the number of copies that should be kept within the cluster. Following our distribution algorithm, the copies of each resource data are placed on the first LR in count nodes from the permutation (of nodes) mapped to the respective resource (Formula 5).

E. *Use-cases*

This section describes Cloud Information Service related use-cases and our sequence of actions in their handling.

Use-case1: Resource Lookup Query

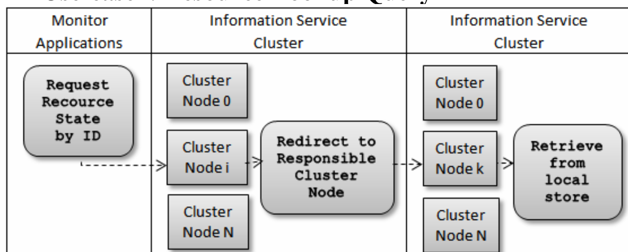

Figure 3. Resource Lookup Query sequence.

The Consumer of resource information requests a resource state by given id, contacting random node in the Information Service Cluster. The node receiving the request, redirects the call to the currently responsible node for that resource. The responsible node retrieves the data from its local store.

Use-case 2: Massive Searching/Listing of Resources

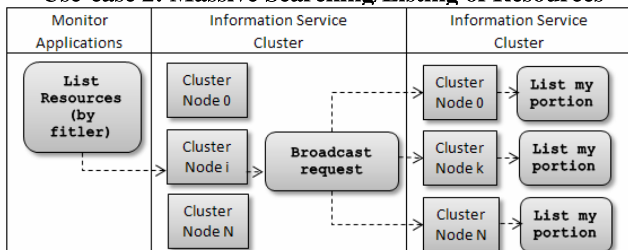

Figure 4. Massive Query Listing sequence.

The Consumer of resource information requests a list of all resources within the datacenter, potentially filtered by some criteria. The request is passed to a random cluster node, which broadcasts it to all other (live) nodes. Every node in the cluster retrieves from its local store the subset of resources, on which he is current owner that satisfy the supplied filtering criteria.

Use-case 3: Resource Information Update

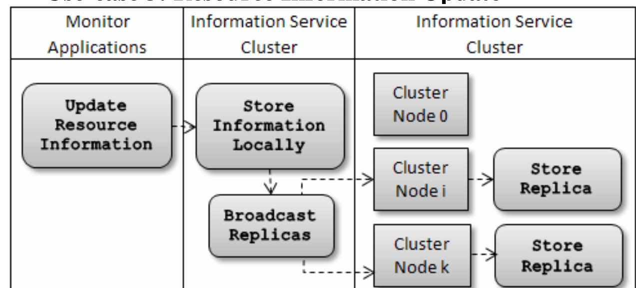

Figure 5. Resource Information Update sequence.

The Producer of resource information sends resource status update to the responsible node for the target resource. The responsible node stores locally the resource information and sends replicas according the preconfigured level of replication. In Cloud environment, we consider that datacenter worker nodes could be redirected to connect to their currently responsible cluster node. If this is not feasible for some specific situation, the use-case should simply be extended with one redirection step internally in the cluster, similarly to Use-case 1.

F. *Utilizing Non-replica Caches*

Information Systems ambitious to provide efficient management of resource data, and respectively – a competitive system performance, usually need to implement runtime caching in order to minimize the drawbacks from slow disk I/O operations. Implementing runtime caches in distributed systems is usually a complicated task, since consistency of the cached data must be maintained via synchronization messages or common access to a shared memory. One of the major advantages of DHTs is the single-place responsibility for a given resource at any moment in time. Thus, our Information Service can abandon the performance dropping complexity of maintaining distributed caches and can use non-replica caches having the guarantee that data will not be modified from different places.

IV. PROTOTYPE AND TECHNOLOGY

We implement a prototype of the proposed system in the Java programming language. Although Java byte code running in a JVM is considered less performant than natively compiled components, we consider it works fine for our purposes. Since we will illustrate the system efficiency when it scales to increasing number of cluster nodes, the fixed performance of each individual node is not of importance to us. Our communication is a custom implemented message-passing-like protocol on the top of Java TCP sockets. For local storage on each node, we decided to use a MySQL database. MySQL was chosen for two reasons. First, it is the most popular free database, it is vastly used and is not

Copyright (c) IARIA, 2011. ISBN: 978-1-61208-171-7

expected to show any eccentric behavior that biases the results. And second, MySQL is also employed in one of the famous grid systems - the European Data Grid [14] and their R-GMA implementation. In this sense, we also decided to use the R-GMA data storing model: in R-GMA instances of given resource type are stored in a dedicated table with table structure (columns) corresponding to the attributes of this resource type. There is one table entry for every resource instance and the entry fields (columns) hold the values of the underlying resource instance attributes [4]. For our experiment, we created one table corresponding to an example type of resource, see Table I.

TABLE I. EXAMPLE STRUCTURE OF A MONITORED RESOURCE TYPE

SAMPLE_RESOURCES				
Resource_id :varchar	Hash :int	Str_value :varchar	Num_value :int	Blob_value :blob

V. RESULTS

A. Testbed Setup

Experiment was performed with six machines (Intel Core 2 Duo E4600 2.4 GHz, 2GB RAM), which we used to form clusters of different sizes – 1, 2, …, 6. All machines were connected through a 100Mbps Ethernet LAN. The software equipment was: Linux *Debian 2.6.18.dfsg.1-12etch2*; Java 6 *update 26;* and MySQL *5.0.32-Debian_7etch6-log*. Client workloads were generated on a laptop (Intel Core 2 Duo 1.73 GHz, 1G RAM) with Windows *XP* and Java 6 *update 26*.

B. Metrics and System Characteristics

Our experiments are focused on studying the following system characteristics:

- Throughput – the number of processed queries in a unit of time. Our metric is: queries per second.
- Response-time (or Latency) – the time taken to answer a query. Our metric is: milliseconds
- Utilization – the distribution of load over the cluster nodes. Our metric is: ratio of locally processed and redirected queries reported by each node.
- Fault-tolerance – utilization of the cluster nodes under churn (i.e., some cluster nodes get inoperable), and degradation of throughput and response-time.

C. Resource Lookup Query Results (Use-case 1)

In this use-case, monitoring applications retrieve resource states by given resource id from the system. To study this scenario, we preliminary loaded the databases on all cluster nodes with volume of one million records - as if there were one million resources of this type (‘*sample_resources*’) throughout the Cloud datacenter. We added identical set of records on all nodes - as if the system LR (level of replication) was set at maximum. During a 10 minute period, simulated “users” submitted blocking queries to the system, waiting for 1 second between successive queries. Client connections were evenly spread to all cluster nodes. The resource ids picked up by ‘users’ were chosen in a stochastic manner. We compared the system scalability with increasing number of users and increasing size of the cluster.

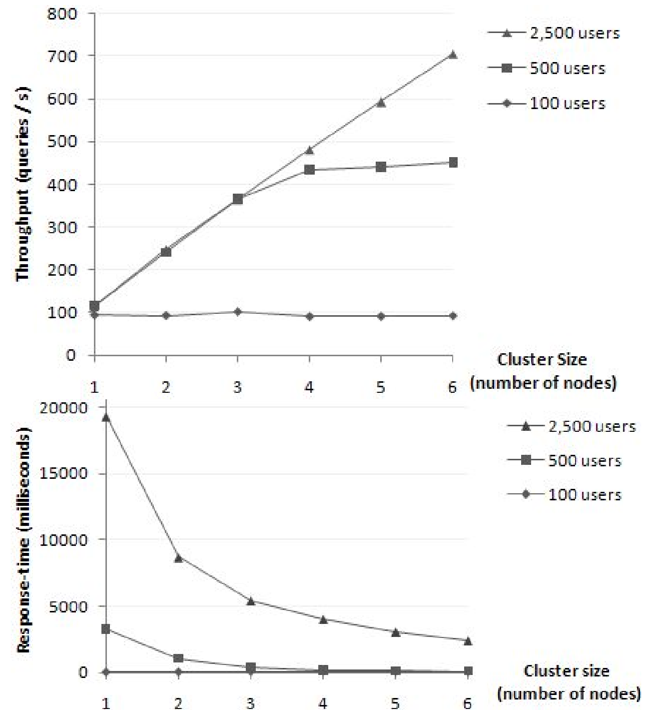

Figure 6. Resource lookup query scaling.

The results shown on Figure 6 illustrate that for a sufficient number of users, i.e., when the system is pushed at its limits, the system throughput increases linearly. For 100 and 500 users the system quickly reaches the maximum, limited by the insufficient client load. From user perspective, the speed-up in response time also improves linearly.

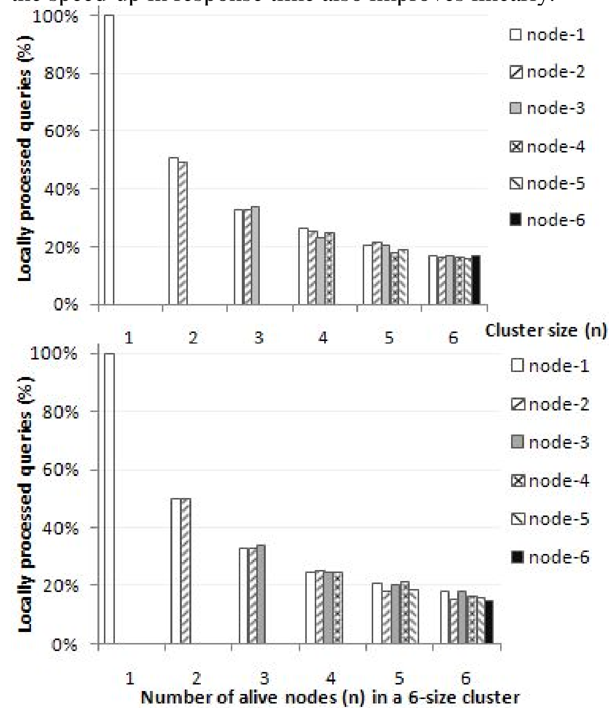

Figure 7. Utilization of cluster nodes.

Copyright (c) IARIA, 2011. ISBN: 978-1-61208-171-7

Figure 7 illustrates the utilization of participating cluster nodes and the real benefit of our DHT keyspace distribution. First, we compared the reported rates of locally processed and redirected calls by each node (see sequence on Figure 3) for different cluster sizes. The results show that all nodes process similar percentage of the received requests locally. We also compared these rates in the fault-tolerance scenarios using a fixed cluster size of 6 nodes and different number of non-operable ones. Results show that for any state of failover rebalancing, the alive nodes are evenly utilized again. We also checked (but are not placing diagram here) that throughput and response-time for fault-tolerance cases, report the same rates as if there was a healthy cluster formed of the respective number of alive nodes.

We made one more experiment for the Resource Lookup Query use-case. To show the benefit of utilizing non-replica caches, we defined cache buffers on all nodes of fixed size - 100,000 entries. With a fixed volume of 1 million resources (of this resource type) in the whole datacenter, the cache-hit rates change with growing of the cluster as follows: 1 node – 10% cache-hit-rate, 2 nodes – 20%, 3 nodes – 30 %, and so on. By running our measurements again, we get a super-linear growing of the throughput as shown in Figure 8.

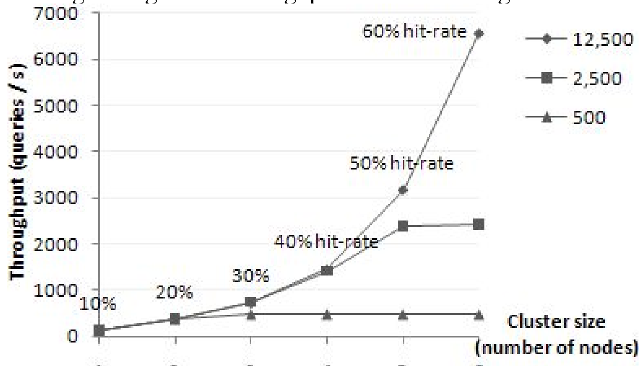

Figure 8. Resource lookup query scaling.

D. *Massive Searching/Listing of Resources (Use-case 2)*

Since these are more rarely triggered queries, used in result of reservation/allocation cases or in global system monitoring and maintenance, we only measured the response-time speed up using one-client load. The same data volume of 1 million resources was used for this experiment.

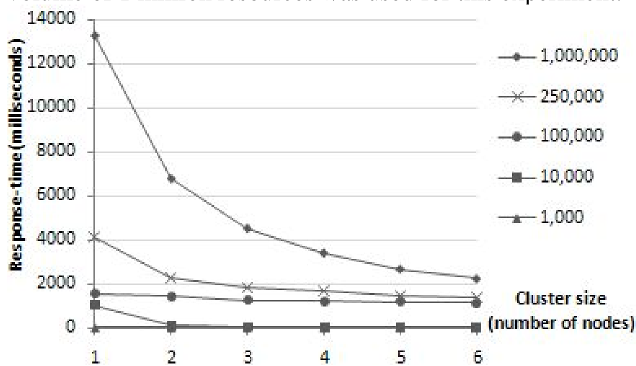

Figure 9. Massive queries scaling.

Figure 9 illustrates the measured speedup for different volumes of the query result set. We used modulo functions upon the 'Hash' field in the SQL where-clause to restrict the proper subset of resources listed by each cluster node (recall that every node also holds replicas owned by other nodes). The exact formula will be left beyond the scope of this paper to not overburden the exposition of experiments.

Again we measured the utilization of cluster nodes. Results on Figure 10 show that all nodes retrieve equal subsets of resources for any cluster size, as well as for any cluster state including fault-tolerance rebalancing. We also checked (but are not placing diagram here) that response times in the fault-tolerance cases remain the same as if there was a healthy cluster formed of the respective alive nodes.

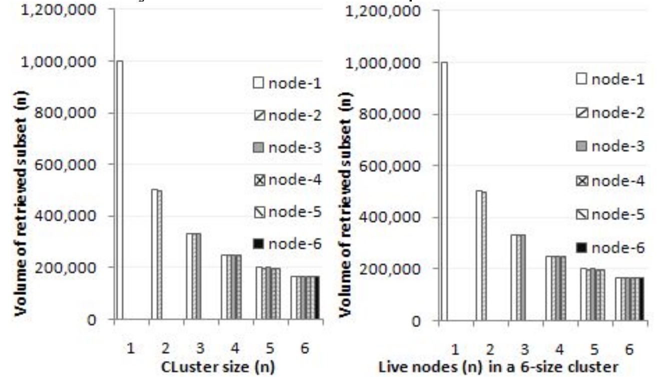

Figure 10. Utilization of node for massive queries.

E. *Resource infromation Update (Use-case 3)*

Similarly to Use-case 1, for this scenario users submitted blocking queries to the system during a 10 minute period, while waiting for one second between successive calls.

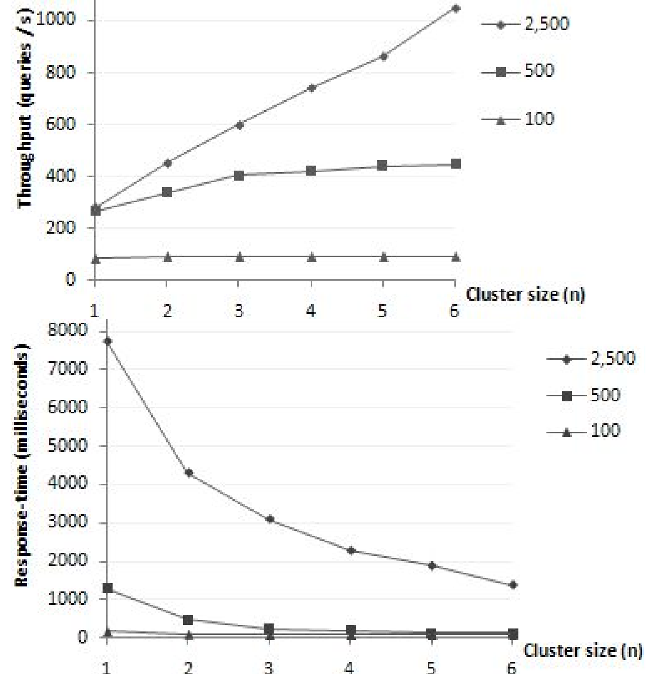

Figure 11. Resource infromation update scaling.

We produced the results with fixed level of replication LR=3, since it is usually considered as best balanced between reliability of data and performance of the system. Results shown on Figure 11 are as expected and again close to the linearity. The particular choice of a replication method usually reflects strongly upon the system performance. In our case, we have chosen to buffer replicas in portions and push them into the databases within grouped transactions. This method performs much faster than storing of the original copies, but makes replicas to appear with a few seconds latency. Both are common effects in replica-maintaining systems. On Figure 12, we fairly illustrate the actual draw-back we get from our case-specific replication method.

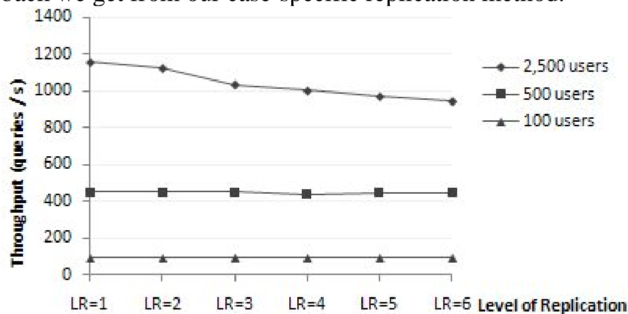

Figure 12. LR in 6-nodes cluster: degradation of throughput.

VI. CONCLUSION AND FUTURE WORK

Our Information Service approach provides important advantages, but strong limitations in the same time. First limitation comes from our DHT-based balancing, which normally requires employment of homogeneous cluster. The second limitation comes from our specific keyspace partitioning algorithm – we cannot easily add new nodes to the cluster, because the whole set of resources should be totally re-balanced. Notice that global rebalancing is not needed when existing nodes from the cluster die and come up again. We consider those limitations completely acceptable for the Cloud resource management. Dedicating a homogeneous cluster when building a farm of computers is not an obstacle; and growing of the cluster is usually related to extending of the physical datacenter, thus being a planned task in long terms. Extending of the cluster then should be done with a dedicated data migration procedure. In trade-off for these limitations, we get advantages that are of major importance for competitive systems as Clouds pretend to be. First, we overcome some major disadvantages of existing Grid systems imposed by the centralized or hierarchical organization. The proposed system combines benefits from the centralized and decentralized organizations, being centric-oriented, and scalable and failover-capable in the same time. The DHT-based balancing ensures performance efficiency in retrieval of resources with no more than one hop redirection. We also showed that utilizing non-replica caches enables Cloud manufacturers to achieve super-linear growing of system throughput via horizontal scaling, i.e., by employing more cluster nodes with enabled RAM buffer caches. Major improvement was also achieved in the fault-tolerance rebalancing in comparison to the traditional DHTs. In traditional DHTs failing of a node causes its 'orphaned' portion of elements to be handled by one or two of its neighbors (see Figure 1). This ends up in uneven load over the nodes left alive. The proposed algorithm ensures equal utilization of the alive nodes for the failover rebalancing, preventing overloaded nodes to become a system bottleneck.

Our future researches will be concentrated on analytical and simulation modeling of the system. We must also find the limits of growing of our cluster, having in mind that open connections are every-to-every. Effort should be spent in studying a modified system with introduced level of vicinity.

REFERENCES

[1] B. Yang and H. Garcia-Molina. Designing a super-peer network, 19th International Conference on Data Engineering (ICDE), (Bangalore, India), Mar. 2003.

[2] B. Plale, P. Dinda, and G. Laszewski. Key Concepts and Services of a Grid Information Service. ISCA 15th International Parallel and Distributed Computing Systems (PDCS), 2002

[3] R. Ranjan, A. Harwood, and R. Buyya. Peer-to-peer based resource discovery in global grids: a tutorial. IEEE Commun Surv Tutorials, 10(2), pp. 6–33, 2008

[4] R-GMA System Specification Version 6.2.0: http://www.r-gma.org/documentation/specification.pdf, 21.09.2011

[5] D. Thain, T. Tannenbaum, and M. Livny. Condor and the Grid. Grid Computing: Making the Global Infrastructure a Reality, John Wiley & Sons, NJ, USA, 2003.

[6] J. Yu, S. Venugopal, and R. Buyya. Grid market directory: A web and web services based grid service publication directory. The Journal of Supercomputing, 36(1), pp. 17–31, 2006.

[7] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski,W. Smith, and S. Tuecke. A directory service for configuring high-performance distributed computations. 6th IEEE Symp. on High Performance Distributed Computing, pp. 365–375. IEEE CS Press, 1997.

[8] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01), Washington, DC, USA, 2001. IEEE CS.

[9] F. Sacerdoti, M. Katz, M. Massie, and D. Culler. Wide area cluster monitoring with ganglia. 5th IEEE International Conference on Cluster Computing (CLUSTER'03), Hong Kong.

[10] D.S. Milojicic, V. Kalogeraki, R. Lukose, and K. Nagarajan. Peer-to-peer computing. Technical Report HPL-2002-57, HP Labs, 2002.

[11] G. Manku. Dipsea: A Modular Distributed Hash Table. Ph. D. Thesis (Stanford University), Aug. 2004.

[12] J. Li, J. Stribling, T. Gil, R. Morris, and M. F. Kaashoek. Comparing the performance of distributed hash tables under churn, 3rd International Workshop on Peer-to-Peer Systems, Feb. 2004

[13] X. Zhang, J. Freschl, and J. M. Schopf, A performance study of monitoring and information services for distributed systems, 12th IEEE International Symposium on High Performance Distributed Computing (HPDC-12), 2003.

[14] The DataGrid Project: http://eu-datagrid.web.cern.ch/eu-datagrid, 21.09.2011

[15] S. Mossige. Generation of permutations in lexicographical order, , pp. 74-75, BIT , ISSN 1572-9125, Vol. 10 (1. 1970).

[16] D. Knuth. The Art of Computer Programming, volume 3 (1973), Sorting and Searching pp. 506–542.

[17] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. 16th international conference on Supercomputing, pp. 84–95, NY, USA, 2002.

[18] P. Ganesan, B. Yang, and H. Garcia-Molina. One torus to rule them all: Multidimensional queries in P2P systems. In Proc. of WebDB, pp. 19–24, 2004.