Resource Adaptive Distributed Information Sharing

Hans Vatne Hansen[1], Vera Goebel[1], Thomas Plagemann[1], Matti Siekkinen[2]

[1] University of Oslo, Department of Informatics
{hansvh|goebel|plageman}@ifi.uio.no
[2] Aalto University School of Science and Technology, Department of Computer Science and Engineering matti.siekkinen@tkk.fi

Abstract. We have designed, implemented and evaluated a resource adaptive distributed information sharing system where automatic adjustments are made internally in the information sharing system in order to cope with varying resource consumption. Our results show that resource adaptation is an efficient way of improving query throughput, and that it is most effective on our hardware when the number of stored data elements is larger than 1000 or when more than 100 queries are run per minute. We have been able to improve the number of performed queries with a factor of 2.5 when 2000 data elements are stored and 100 concurrent queries are issued per minute, and with a factor of 4.7 when 1000 data elements are stored and 250 concurrent queries are issued per minute.

Keywords: autonomic networks, self-optimization, resource adaptation.

1 Introduction

Data centric networking has become an important networking paradigm. Indeed, focus on content instead of its location has started to drive the design of communication systems, and over the last decade much research efforts have been invested into developing solutions for networking data instead of hosts. These activities have addressed several kinds of approaches ranging from building overlay type of solutions[10, 7, 5] to designing complete data-centric network architectures from scratch[4, 2].

It would be desirable to have generic information sharing solutions that are capable of storing and looking up various kinds of data and meta-data. This is true for both the current situation as well as for the future. Specific applications today often rely on particular overlay solutions and the different solutions are usually non-interworking, which introduces significant amount of unnecessary overhead and complexity. Futuristic Internet architectures such as *ANA (Autonomic Network Architecture)* would benefit from such a solution as an integral and reusable core component of the architecture. To this end, we have designed and developed a fully distributed system called *MCIS (Multi-Compartment Information Sharing)* which is based on a *distributed hash table (DHT)* type of structured peer-to-peer system. The basic idea is that within the entire system, each different data type forms a separate data compartment. A data compartment represents a logical unit for data management, like network compartments or realms form logical network units. Data items and queries are in each data compartment routed independently of other data compartments. Furthermore, the system supports multi-attribute data and range queries.

One of the challenges in deploying such a fully distributed system that relies on cooperation of every node in looking up and storing data is that these nodes can have significantly different amounts of resources available, such as CPU, storage space, or energy. The amount of available resources depends on the one hand on the device itself, e.g. mobile devices have scarce resources compared to desktop computers, and on the other hand, on the current workload caused by MCIS and other services and applications running on that node. A single overloaded node can negatively impact the performance of the overall system. Thus, there are potentially several bottlenecks in the system in form of overloaded nodes. Therefore, in order for such a system to perform well, nodes should avoid overload situations by dynamically adjusting the resource consumption imposed by the information sharing system.

In this paper, we describe the design of MCIS enhanced with such an automated resource adaptation mechanism. MCIS is designed in such a way that for handling multi-attribute data there are so called hubs created separately for each attribute which route data and queries independently. These hubs introduce replication of data in order to gain performance in query processing. We take advantage of this design in the resource adaptation mechanism so that there is a trade-off in the level of replication and the query processing efficiency, against resource consumption and the number of active hubs. We introduce a utility metric to identify those hubs where leaving has the most positive effect on the query processing efficiency.

We evaluate the performance and behavior of the resource adaptive MCIS by storing and retrieving real Internet traffic traces from CAIDA[12]. The evaluation results do not only demonstrate the feasibility of resource adaptation in MCIS, but also the superior performance of a resource adaptive MCIS compared to a non-adaptive MCIS. This work is done as part of the ANA project and the MCIS source code is available at the project website (http://ana-project.org/).

The remainder of this paper is structured as follows. Section 2 introduces MCIS. A detailed description of the resource adaptation scheme is given in Section 3. In Section 4, we evaluate the system behavior. Finally, in Section 5, we draw conclusions and explaining future work items.

2 Multi-Compartment Information Sharing

MCIS is a fully distributed store and query system. It is based on Mercury[1] which is a DHT-based system that supports multi-attribute range queries, such as `size < 100 MB`, `type = MPEG` and `name = *` when looking up MPEG video content with any name and size smaller than 100 MB. As a DHT-based system, MCIS is fully decentralized.

Each different data type is a separate logical data compartment formed by an individual instance of a Mercury system. A data type is represented as several attribute value pairs like *int size = 37*, *string type = AVI* and the specification of the collection of these attributes is called a schema. Each data type forming a data compartment consists of attribute hubs which are, similarly to Chord, logical ring structures, one per attribute. These hubs organize, manage, and route the data and queries related to the specific attribute independently from each other as follows. Data elements are replicated and inserted into all hubs while a query is only forwarded to the hub where it is expected to be executed most efficiently, which depends on the specified ranges for each attribute. For example, a wild card attribute implies that the query would need to be evaluated at each node in the hub corresponding to that attribute, while a small range for another attribute narrows the search down to only few nodes in that attribute hub. In this way, replication is introduced for the sake of efficiency of query processing. Fig. 1 shows how a data item is inserted into all attribute hubs and how a potential query could retrieve the data from one of the hubs.

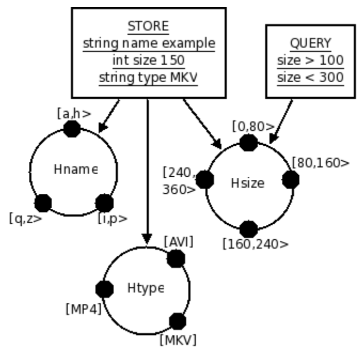

Fig. 1. MCIS hub structure

It should be noted that hubs do not necessarily need to be actively used for all the attributes in the schema, but they help to decrease hop count in a system with a large number of nodes. In other words, hubs can be joined or left at any time in order to reduce resource consumption, but with the expense of potentially heavier query processing and longer query response times. This is the trade-off our resource adaptation mechanism relies on. Furthermore, not every node of a specific data compartment needs to participate in each active attribute hub. As shown in Fig. 1, there are four nodes in the system. All of these nodes participate in `Hsize`, but only three of them participate in the two other hubs.

On top of the data compartments there is a special meta-data compartment, also a Mercury instance, which enables looking up specific data compartments.

This meta-data compartment is the glue that binds the different data compartments together. For example, when a node wishes to lookup data of a specific type, it makes a lookup in the meta-data compartment with attribute constraints specifying the data type. As a result, the node will receive the indentifiers of the matching data compartments.

3 Resource Adaptation

Distributed systems like MCIS can have a high degree of churn, and one reason for this is failing nodes[8]. There is a risk that one failing node can harm the entire system, and even with data replication, the overall performance decreases with node failures. The main problem in distributed systems is that the number of simultaneous users becomes too high for the system to handle and that this becomes a bottleneck. Our approach to solve this problem is with self-optimization, which in case of MCIS means to automatically detect certain changes in the individual nodes and adapt to these changes in order to improve the service.

The amount of available resources for MCIS changes continuously. Available memory and processing power are critical system resources that vary depending on all running processes on the machine. A possible strategy for self-optimization is to identify when resource consumption is at a level where MCIS is unable to function properly or to service its users, make internal changes to adapt to this, and consequently improving performance. It is also possible to adapt to variance in other resources as well, such as storage space or energy. We call this resource adaptation. It consists of two distinct functions in addition to the actual MCIS application:

- The resource consumption is measured in order to determine when the consumption is at a critical level. We focus on CPU load, but any other node resource could be measured.
- The measurement data is analyzed, and attribute hubs are joined or left when thresholds are reached.

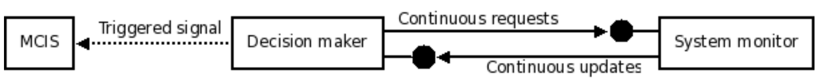

Fig. 2. Feedback control system

Fig. 2 shows our feedback control system for resource adaptation of MCIS. The two resource adaptation modules dictate how many attribute hubs each data compartment in MCIS has, based on CPU load and pre-defined thresholds.

The **System monitor** inspects system resources at fixed intervals and makes the obtained information available to the **Decision maker**. We choose processor load as the resource adaptation trigger, because calculating query routes in the attribute hubs is a CPU intensive task. Without sufficient available CPU capacity, the queries will not be issued. The load is calculated based on CPU queue length, i.e. the number of processes in the waiting queue. We normalize this measure by factor 100 such that an idle computer has a load of 0 and a fully utilized CPU has a load of 100.

The adjustment made in each MCIS node is whether an attribute hub should be joined or left. This is not a coordinated event, but made on a per machine basis. Individual nodes can leave a hub while the remaining nodes can continue to use it if they have enough resources. Leaving an attribute hub on a machine leads to less routing calculation, but could potentially lead to more hops. By minimizing calculations of where to retrieve data we expect that the MCIS node is able to answer more queries. This kind of optimization is especially valuable for resource constrained devices like PDAs or systems where resource demanding applications use a large percentage of the available CPU.

We choose which hub to join and leave based on what we call a *utility score*. The utility score ranks the different hubs based on how profitable they are when performing queries. Data elements are always replicated and sent to each hub, but queries are only forwarded to the most selective hub, which is then responsible for providing results. Finding the optimal hub can therefore minimize routing and improve performance. MCIS calculates the utility score for each hub in the following way: If a hub stores a data element, its utility score is decreased by one, but if it answers a query, the score is reset to zero. The result of this formula is that hubs that answer many queries will have a utility score close to zero, while less utilized hubs have negative scores. When MCIS is told to leave a hub, it chooses the hub with the lowest score. The reasoning behind this strategy is to first get rid of those hubs that add to resource usage by only routing data items, but never route queries.

Hub	T0	T1	T2	T3	T4
`Hname`	0	-3	-3	-5	0
`Hsize`	0	-3	0	-2	-2
`Htype`	0	-3	-3	-5	-5

Table 1. Example time-line of utility scores

Table 1 shows an example of how utility scores would be calculated in a given situation. At T1, 3 data elements are stored and all the hubs decrease their score. At T2, a query is forwarded to `Hsize` making it reset its score to 0 while the other two remain -3. At T3, 2 new data elements are stored. At T4, a query is forwarded to `Hname` giving the three hubs three different utility scores. The most important hub at this point in time is `Hname`. `Hsize` is less important, while `Htype` is least important and will be the first hub an overloaded node leaves.

4 Evaluation

The objective of our evaluation is to investigate the differences between MCIS with and without self-optimization and see if MCIS can perform better when adapting to system resource consumption in form of CPU load. The desirable outcome of our evaluation is higher efficiency and an increased number of queries performed when resource adaptation is applied. The number of simultaneous queries that MCIS can perform is known as throughput rate and is expressed as queries per time unit. We choose to evaluate our system based on throughput as it reflects the demands and requirements that applications using MCIS have for performance.

The number of hubs in MCIS is one of the key parameters in our studies. It varies automatically according to resource consumption and influences how query routing is performed. External load is also an important parameter in our evaluation. We use synthetic load to have total control over the span and when the load is applied. The external load is generated by two programs with the purpose of using a predetermined amount of CPU.

We aim at keeping our evaluation realistic and choose data and queries that represent real world use-cases. The data we use in our studies are real Internet traffic traces gathered by CAIDA[12], and a typical investigation on these traces is anomaly detection. One example of our queries is locating traffic on a range of ports, known to be used by malicious programs. These queries also invoke route calculations and might not be issued if the CPU load becomes too high.

The parameters we vary are the number of *inserted data elements (DE)* and the number of *queries per minute (QPM)*. We choose these parameters because they correspond well to the load of MCIS. In our experiments the data elements are stored in MCIS before the queries are performed. By changing one or both of these parameters we can investigate the implications and understand in which situations, if any, resource adaptation can improve MCIS.

We conduct our evaluation experiments in two phases with one local and one distributed test. In the first test, we use one MCIS node and experiment with a wide range of parameters. This test is an attempt to narrow our parameter values and prepare for the second, distributed test.

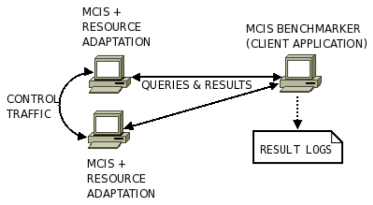

Fig. 3. Distributed evaluation setup

We use the same configuration on all the machines in our experiments to ensure that queries are processed under the same conditions, independently of which MCIS node they are forwarded to. The machines are Intel Pentium 4s with 2.60 GHz and 1 GB of memory. In the distributed test, all the machines are connected directly to each other through the same switch. There are two MCIS nodes and one client node running, as shown in Fig. 3.

In the **local test**, we find that MCIS uses an average CPU load of 5 on the given hardware. This means that if the CPU load is below 95, it is possible for MCIS to get sufficient CPU time. We conclude that a CPU load of 95 should be the trigger for leaving an attribute hub as this is the critical level where MCIS can function properly. With different hardware specifications, the CPU requirements will differ and the triggers must be adjusted accordingly. However, the same concepts apply. We determine all our parameter values for inserted data elements and queries per minute based on what MCIS can handle on our machines.

	MINIMUM	**MAXIMUM**
Data elements	1000	2000
Queries per minute	100	300
External CPU load	100	100
Trigger thresholds	95	100

Table 2. Parameter values

Table 2 shows our findings in the local test and what parameter value ranges we use in the **distributed test**. We perform one experiment for each of the varying query rates and number of inserted data elements with resource adaptation disabled, and with resource adaptation enabled.

Throughput in MCIS is the number of performed queries, and it represents the overall performance of the system. When comparing throughput between the experiments with and without resource adaptation, it measures how well the self-optimization works.

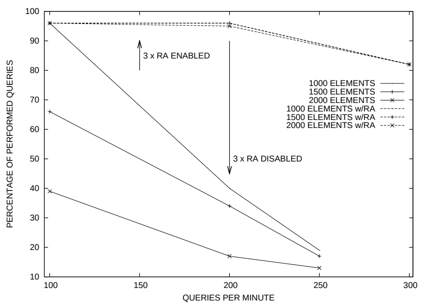

Fig. 4. Throughput comparison

Fig. 4 shows that the experiments are almost equal when resource adaptation is enabled, regardless of the number of stored data elements. MCIS is able to perform close to 100% of all the queries when both 100 and 200 queries are made per minute. This is not the case when adaptation is disabled. In the experiments where resource adaptation is disabled, the number of queries performed drops drastically when the rate is increased from 100 queries per minute to 200 queries per minute. This is especially true for the experiments with 1000 and 1500 stored data elements. In addition, we are unable to achieve consistent results when trying to query MCIS without adaptation 300 times per minute because the machines are fully saturated. Therefore, we decrease the maximum query rate to 250 when resource adaptation is disabled.

Our analysis shows that resource adaptation has a positive effect on throughput and that it significantly improves the performance of MCIS. Most improvement is achieved when the number of stored data elements is large or when query rate is high. As expected, when a sufficient amount of resources are available the effect of resource adaptation is neglectable.

5 Related Work

Several scalable information systems that gather information about networked systems have been proposed in the literature such as in [11, 13, 6]. The first two approaches focus more on information aggregation aspects, while the last approach exhibits some self-configuration properties regarding resource utilization. InfoEye[6] is a self-configuring distributed information management system where management nodes communicate with monitoring sensors located at each overlay node in order to get information about the overlay nodes which execute application tasks. This information is provided via a query interface to applications. Since collecting information about every attribute of each overlay node is unfeasible in a large system, InfoEye self-configures in an optimal fashion the way it gathers the information from monitoring sensors, e.g. which attributes, from which nodes, and via push or pull mechanism. InfoEye with the centralized management nodes (one such node exists in [6]) is a quite different concept from MCIS which is a fully distributed system in which nodes make independent decisions.

Replication in distributed systems has been studied earlier. For example, a popularity/size-adaptive object replication degree customization scheme is described in [14]. In [3], the focus is on maximizing object availability in peer-to-peer communities under space constraints. Together with an additional large body of approaches, this work focuses on developing optimal models for replication in a system given certain constraints and objectives. Our scheme differs in that there is no system wide coordination. Instead, individual nodes dynamically and independently make adjustments depending on their system state.

The closest match to our work that we could find is the concept of *Elastic Routing Table (ERT)*[9]. This mechanism is proposed to prevent overload situations at particular DHT nodes due to inherent load balancing problems in

DHTs, the heterogeneity of network nodes, and the non-uniform and time varying popularity of content. ERT uses variable size routing tables for DHT nodes which are adjusted based on load at the particular node. The difference to our work is that we consider impacting the replication degree of data instead of the routing tables. In fact, ERT could be used as a complementary scheme in MCIS to adjust each node's routing tables for individual hubs.

6 Conclusions

In this paper, we presented a generic distributed information sharing system which is able to dynamically adapt to resource consumption when necessary. This property allows the system to be deployed among heterogeneous nodes while alleviating the problem of individual nodes to become bottlenecks, which would hurt the performance of the entire system. Our evaluation results demonstrate that with the use of resource adaptation, the system achieves higher throughput than without resource adaptation. These results are also relevant for other distributed systems, not only MCIS.

This paper presents our first results in our pursue towards a fully self-organizing information sharing system. Hence, there are a number of challenges that we would like to address in our future work. The current version of MCIS attempts to avoid overload situations of individual nodes. We want to benefit from the large body of analytical work on replication for high availability in order to drive the design of MCIS towards optimal resource adaptation. This work also includes replicating data through duplicate attribute hubs, which requires changes to the design of the underlying Mercury system. In addition, we plan to model the resource consumption to be able to evaluate the behavior of the MCIS system with a large number of nodes through simulations. Specifically, we are interested in understanding the quantitative impact of nodes leaving and joining hubs to the query response time in a large system. We are also interested in analyzing the trade off between the number of joined hubs and query hop count.

References

1. Ashwin R. Bharambe, Mukesh Agrawal, and Srinivasan Seshan. Mercury: Supporting scalable multi-attribute range queries. *ACM SIGCOMM*, 2004.
2. Petri Jokela, András Zahemszky, Christian Esteve Rothenberg, Somaya Arianfar, and Pekka Nikander. Lipsin: line speed publish/subscribe inter-networking. In *SIGCOMM '09: Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pages 195–206, New York, NY, USA, 2009. ACM.
3. J. Kangasharju, K.W. Ross, and D.A. Turner. Optimizing file availability in peer-to-peer content distribution. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 1973–1981, May 2007.
4. Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A data-oriented (and beyond) network architecture. *SIGCOMM Comput. Commun. Rev.*, 37(4):181–192, 2007.

5. X. Li, F. Bian, H. Zhang, C. Diot, R. Govindan, W. Hong, and G. Iannaccone. MIND: A distributed multi-dimensional indexing system for network diagnosis. In *Proceedings of INFOCOM 2006*, pages 1–12, 2006.
6. Jin Liang, Xiaohui Gu, and Klara Nahrstedt. Self-configuring information management for large-scale service overlays. In *INFOCOM*, pages 472–480, 2007.
7. Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *Proceedings of SIGCOMM '01*, pages 161–172, 2001.
8. Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling churn in a dht. In *ATEC '04: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.
9. Haiying Shen and Cheng-Zhong Xu. Elastic routing table with provable performance for congestion control in dht networks. In *ICDCS '06: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, page 15, Washington, DC, USA, 2006. IEEE Computer Society.
10. Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of SIGCOMM '01*, pages 149–160, 2001.
11. Robbert Van Renesse, Kenneth P. Birman, and Werner Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Trans. Comput. Syst.*, 21(2):164–206, 2003. http://doi.acm.org/10.1145/762483.762485.
12. Colby Walsworth, Emile Aben, kc claffy, and Dan Andersen. The caida anonymized 2009 internet traces - equinix-chicago.dira.20090331-055905.utc. http://www.caida.org/data/passive/passive_2009_dataset.xml.
13. Praveen Yalagandula and Mike Dahlin. A scalable distributed information management system. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 379–390, New York, NY, USA, 2004. ACM.
14. Ming Zhong, Kai Shen, and Joel Seiferas. Replication degree customization for high availability. In *Eurosys '08: Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*, pages 55–68, New York, NY, USA, 2008. ACM.