# 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 our information sharing system in order to cope with varying resource consumption. CPU load is monitored and a light-weight trigger mechanism is used to avoid overload situations on a per-machine basis. Additional improvements are obtained by calculating what we call a utility score to better determine how the data structures in the system should be arranged. Our results show that resource adaptation is an efficient way of improving query throughput, and that it is most effective when the number of stored data items in the system is large or many queries are performed concurrently. By applying resource adaptation, we are able to significantly improve the performance of our information sharing system.

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

## 1 Introduction

Data centric networking has become an important networking paradigm. 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 [13, 10, 7] to designing complete data-centric network architectures from scratch [6, 4].

It is desirable to have general purpose information sharing solutions that are capable of storing and querying various kinds of data. This is true for both the current situation as well as for the future. Many applications today rely on specific overlay solutions, and the different solutions are usually incompatible, which introduces significant amount of unnecessary overhead and complexity. Future Internet approaches such as ANA (Autonomic Network Architecture) [1] would benefit from such a general purpose information sharing system as an integral and reusable core component of the architecture. We have designed and

implemented 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 core functionality of MCIS is storing and querying different types of data. We define a data type as a set of attributes and a data item as a tuple of attribute values. Within the entire MCIS system, each different data type is formed by separate, logical units for data management called attribute hubs. In these attribute hubs, data items and queries are routed independently from each other. Furthermore, the system supports multi-attribute range queries.

One of the challenges in deploying such a fully distributed system that relies on cooperation of every node, is that these nodes can have significantly different amounts of resources available, such as CPU, memory, storage space and 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 the 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, individual nodes should handle overload situations by dynamically adjusting to the varying resource consumption.

In this paper, we describe our MCIS system enhanced with an automated resource adaptation mechanism. MCIS is designed in such a way that the different attribute 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 attribute hubs. We introduce a utility metric to identify the hubs with the most positive effect on the query processing efficiency and give them priority when reducing the number of active hubs.

We evaluate the performance and behavior of the resource adaptive MCIS by storing and retrieving real Internet traffic traces from the Cooperative Association for Internet Data Analysis (CAIDA) [15]. 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 complete source code of our system 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. In Section 5, we draw conclusions and explain future work.

## 2 Multi-Compartment Information Sharing

MCIS is a fully distributed store and query system. It is based on Mercury [2] which is a DHT-based system that supports multi-attribute range queries,

such as `size > 100 MB`, `size < 300 MB` when looking up data items with size between 100 MB and 300 MB. As a DHT-based system, MCIS is fully decentralized.

Each data type in MCIS is administrated by an instance of a Mercury system. The data types can have several attributes like `string name`, `int size`, and `string type`; and the specification of the collection of these attributes is called a schema. An example of a data item for this schema is ‘‘`example, 150, MKV`‘‘. The data types are managed by attribute hubs which are logical ring structures, one per attribute in the schema. These attribute hubs, like `Hname`, `Hsize` and `Htype` in Figure 1, organize and route the data items related to the specific attribute independently of each other. This is done in the following way: Data items are replicated and stored in all hubs while a query is only forwarded to the hub where it is expected to be executed most efficiently. A wild card attribute in a query needs 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 other hub. In this way, replication is introduced to improve query processing efficiency. Figure 1 shows how a data item is stored in all attribute hubs, i.e. `Hname`, `Hsize` and `Htype`, and how a potential query could most efficiently retrieve the data from `Hsize`.

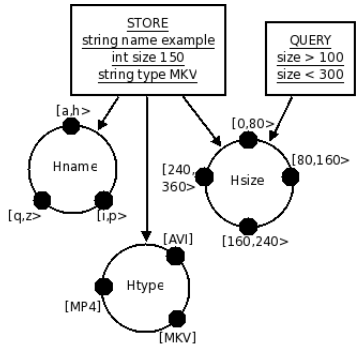


**Fig. 1.** MCIS hub structure

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 adapt to resource consumption, but with the expense of potentially heavier query processing and longer query response times when few hubs are active. This potential expense is the trade-off our resource adaptation mechanism relies on. As shown in Figure 1, there are four nodes in the system. All of these nodes participate in `Hsize`, but only three of them participate in `Hname` and `Htype`.

## 3 Resource Adaptation

Distributed systems like MCIS can have a high degree of churn, and one reason for this is failing nodes [11]. 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 concurrent users can become too high for the system to handle. Our solution to this problem is self-optimization, which in MCIS means to automatically detect certain changes in the individual nodes and adapt to these changes in order to improve the service.

Available memory and processing power are critical system resources that vary depending on all running processes on the machine. Our 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 improve performance. It is possible to adapt to variance in almost all resources such as memory, storage space and energy. We call this resource adaptation, and it consists of two distinct functions in addition to the actual MCIS application: 1) 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. 2) The measurement data is analyzed, and nodes join or leave attribute hubs when pre-defined thresholds are reached.

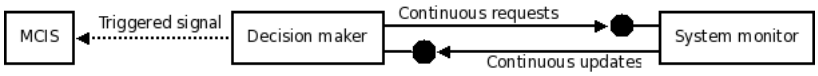


**Fig. 2.** Feedback control system

Figure 2 shows our feedback control system for resource adaptation of MCIS. The two resource adaptation modules dictate how many active attribute hubs each data type 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 use 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 successfully executed. The processor load is calculated based on CPU queue length, i.e. the number of processes in the waiting queue when idle and blocking processes are omitted. This load number is calculated by the Linux kernel [9] as the exponentially weighted moving average within a one minute window. It is found to perform better than utilization indices when doing dynamic load balancing in distributed systems [3]. We normalize this measure by factor 100 such that an idle computer has a load of 0 and a fully loaded CPU has a load above 100. We have also investigated memory utilization, but found that it did not influence performance enough to be used as trigger.

The adjustment made in each MCIS node is whether it should join or leave a certain attribute hub. 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 available resources. Leaving an attribute hub leads to less routing calculation with a potential drawback of extra hops and higher response times. 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 processing capacity.

We choose which hub to join and leave based on what we call a *utility score* which ranks the different hubs based on how profitable they are when performing queries. Data items are always replicated and sent to each hub, but queries are only forwarded to the most selective hub, which is then responsible for providing the results. Hence, leaving the attribute hub where fewest queries are forwarded minimizes routing overhead. The utility score for each hub is calculated in the following way: If a hub stores a data item, its utility score is decreased by one, but if it forwards 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 leave the hubs that add to resource usage by routing data items, but never route queries.

| | 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 are calculated in a given situation. T0 through T4 are points in time, ranging from oldest to newest. At T1, 3 data items are stored and all the hubs decrease their utility score. At T2, a query is forwarded to `Hsize` making it reset its score to 0 while the other two remain at -3. At T3, 2 new data elements are stored. At T4, a query is forwarded to `Hname` giving the hubs three different utility scores. The most valuable hub at this point in time is `Hname`. `Hsize` is less important, while `Htype` is least important and the first hub an overloaded node leaves.

In summary, our resource adaptation extension is a simple and light-weight solution where the introduced overhead is too small to measure correctly. It monitors CPU load and calculates utility scores, but does not require global coordination or introduce extra networking traffic. All changes are made on a per-machine basis, and the potential result is fewer node failures and more queries performed.

## 4 Evaluation

The objective of the evaluation is to investigate the differences between our MCIS system with and without self-optimization to quantify the resulting performance improvement when adapting to changing resource consumption. The desirable outcome of the 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.

In our studies, the number of active hubs in MCIS is one of the key parameters. It varies according to resource consumption and influences how routing is done. External load is also an important parameter, and we use synthetic load to have total control over the quantity and when the load is applied. It is generated by two programs with the purpose of using a predetermined, fixed amount of processing capacity. We aim at keeping our evaluation realistic and choose data items and queries that represent real world use-cases. The data items we use in our studies are real Internet traffic traces gathered by CAIDA [15], and a typical investigation of 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 system has insufficient resources. Other queries have also been tested.

The parameters we vary explicitly are the number of *stored data elements (DE)* and the number of *queries per minute (QPM)*. We choose these parameters because they correspond well to the internal load of MCIS. 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.

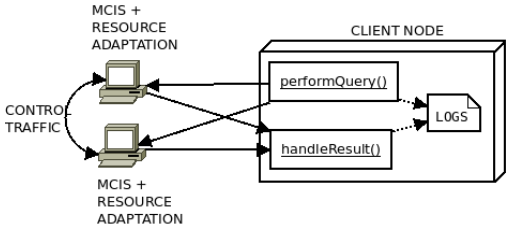


**Fig. 3.** Distributed evaluation setup

We repeat the experiments three times and use the same configuration on all machines to ensure that queries are processed under the same conditions, independently of which MCIS node they are forwarded to. The machines have 2.60 GHz processors 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 Figure 3.

In the **local test**, we see that MCIS has an average CPU load of 5 on the given hardware. This means that if the external load is below 95, MCIS has sufficient CPU. We conclude that a load of 95 can be the trigger for leaving an attribute hub as this is the critical level where MCIS can function properly. With different hardware specifications the load numbers will differ and the triggers must be adjusted accordingly. We determine all our parameter values based on what MCIS can handle on our hardware.

| | Minimum | Maximum |
|---|---|---|
| Data elements | 1000 | 2000 |
| Queries per minute | 100 | 300 |
| External CPU load | 100 | 100 |

**Table 2.** Parameter values

Table 2 shows our results from the local test and what parameter value ranges we use in the **distributed test**. For each of the varying query rates and number of stored data elements, we perform experiments with and without resource adaptation enabled. The improvement of the self-optimization is indicated when comparing throughput between the experiments with and without resource adaptation.

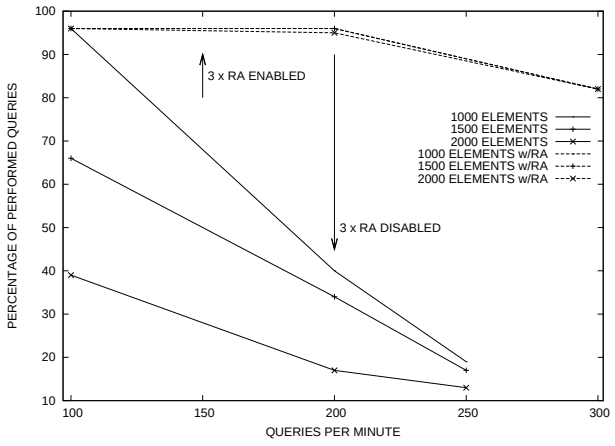


**Fig. 4.** Throughput comparison

Figure 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 performed 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. The machines are fully saturated and we need to 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 [14, 16, 8]. The first two approaches focus more on information aggregation aspects, while the last approach exhibits some self-configuration properties regarding resource utilization. InfoEye [8] 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 [8]) 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 [17]. In [5], 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) [12]. 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 present MCIS, our distributed information sharing system which is able to self-adapt to changing levels of resource consumption when necessary. We achieve self-adaptation by altering internal data structures with the possible side-effect of extra hops and higher response times. This property allows the system to be deployed among heterogeneous nodes while alleviating the problem of individual nodes to become bottlenecks, which can decrease 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 that can trade-off response time for throughput.

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 handle overload situations in 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.

**Acknowledgments.** This work has been funded by the ANA project (EU FP6-IST-27489). The authors would like to thank Stein Kristiansen for valuable feedback.

## References

1. ANA Project. *ANA Blueprint*, sixth framework programme edition, February 2008.
2. Ashwin R. Bharambe, Mukesh Agrawal, and Srinivasan Seshan. Mercury: Supporting scalable multi-attribute range queries. *ACM SIGCOMM*, 2004.
3. Domenico Ferrari and Songnian Zhou. An empirical investigation of load indices for load balancing applications. Technical Report UCB/CSD-87-353, EECS Department, University of California, Berkeley, May 1987.

4. 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.
5. 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.
6. 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.
7. 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.
8. Jin Liang, Xiaohui Gu, and Klara Nahrstedt. Self-configuring information management for large-scale service overlays. In *INFOCOM*, pages 472–480, 2007.
9. The Linux man-pages project. *Linux Programmer's Manual*, November 1997. SYSINFO(2).
10. 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.
11. 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.
12. 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.
13. 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 the ACM SIGCOMM '01 Conference*, pages 149–160, San Diego, California, August 2001.
14. 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.
15. 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.
16. 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.
17. 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.