Towards a Scalable Distributed Information Management System

Praveen Yalagandula and Mike Dahlin
University of Texas at Austin

Abstract

This paper makes a case for developing a general Scalable Distributed Information Management System (SDIMS) abstraction that aggregates *information about large-scale networked systems, provides detailed views of nearby information and summary views of global information, and can serve as a basic building block for a broad range of large-scale distributed applications. This paper outlines the requirements of a SDIMS, describes an initial prototype, lays out key research challenges, and provides directions for tackling those issues.*

1 Introduction

This paper makes the case for developing a general Scalable Distributed Information Management System (SDIMS) abstraction that *aggregates* information about large-scale networked systems and that can serve as a basic building block for a broad range of large-scale distributed applications. The paper then describes our initial efforts for constructing such an abstraction as a peer-to-peer system, focusing on both the promising high-level properties of such an approach and the key research challenges in realizing this goal.

We speculate that a SDIMS in a networked system would provide a "distributed operating systems backbone" and facilitate the development and deployment of new distributed services. For example, monitoring, querying, and reacting to changes in the state of a distributed system are core components of applications such as system management [8, 16], data sharing and caching [12, 14, 17, 19, 21], sensor monitoring and control [10], multicast tree formation [3, 4, 15, 18, 20], and naming and request routing [5, 6].

In these and other large scale information systems, *hierarchical aggregation* is a fundamental abstraction for scalability. Rather than expose all information to all nodes, hierarchical aggregation allows a node to access detailed views of nearby information and summary views of global information. In a SDIMS based on hierarchical aggregation, different nodes can therefore receive different answers to the query "find a [nearby] node with at least 1 GB of free memory" or "find a [nearby] copy of file foo." A hierarchical system that aggregates information through reduction trees [10, 15] allows nodes to access information they care about while maintaining system scalability.

This paper outlines the requirements of a SDIMS, describe an initial prototype, lays out key research challenges, and provides directions for tackling those issues.

2 Requirements

In order to be a general framework, a SDIMS should have four properties. First, the system should accommodate large numbers of participating nodes, and it should allow applications to install and monitor large numbers of data attributes. Enterprise and global scale systems today might have tens of thousands to millions of nodes and these numbers will increase as desktop machines give way to larger numbers of smaller devices. Similarly, we hope to support many applications and each application may track several attributes (e.g., the load and free memory of a system's machines) or millions of attributes (e.g., which files are stored on which machines).

Second, the system should have *flexibility* to accommodate a broad range of applications and attributes. For example, *read-dominated* attributes like *numCPUs* rarely change in value, while *write-dominated* attributes like *numProcesses* change quite often. An approach tuned for read-dominated attributes will suffer from high bandwidth consumption when applied for write-dominated attributes. Conversely, an approach tuned for write-dominated attributes may suffer from unnecessary query latency or imprecision for read-dominated attributes. Therefore, a SDIMS should provide a flexible mechanism that can efficiently handle different types of attributes, and leave the policy decision of tuning read and write propagation to the application installing an attribute.

Third, a SDIMS should provide *autonomy and isolation*. In a large computing platform, it is natural to arrange nodes in an organizational or an administrative hierarchy. A SDIMS should support administrative autonomy so that, for example, a system administrator can control what information flows out of her machines and what queries may be installed on them. And, a SDIMS should provide isolation in which queries about a domain's information can be satisfied within the domain so that the system can operate during disconnections and so that an external observer cannot monitor or affect intra-domain queries.

Fourth, the system must be *robust* to node failures and disconnections. A SDIMS should adapt to reconfigurations in a timely fashion and should also provide mechanisms so that applications can exploit the tradeoff between the cost of adaptation versus the consistency level in the aggregated results when reconfigurations occur.

3 SDIMS Architecture

We draw inspiration from two previous works: *Astrolabe* and *Distributed Hash Tables (DHTs)*.

Astrolabe [15] is a robust information management system. Astrolabe provides the abstraction of a single logical aggregation tree that mirrors a system's administrative hierarchy for autonomy and isolation. It provides a general interface for installing new aggregation functions and provides eventual consistency on its data. Astrolabe is highly robust due to its use of an unstructured gossip protocol for disseminating information and its strategy of replicating all aggregated attribute values for a subtree to all nodes in the subtree. This combination allows any communication pattern to yield eventual consistency and allows any node to answer any query using local information. This high degree of replication, however, may limit the system's ability to accommodate large numbers of attributes. Also, although the approach works well for read-dominated attributes, an update at one node can eventually affect the state at all nodes, which may limit the system's flexibility to support write-dominated attributes.

Recent research in peer-to-peer structured networks resulted in Distributed Hash Tables (DHTs) [3, 4, 6, 12, 13, 14, 17, 18, 19, 20, 21]—a data structure that scales with the number of nodes and that distributes the read-write load for different queries among the participating nodes. It is interesting to note that although these systems export a global hash table abstraction, many of them internally make use of what can be viewed as a scalable system of aggregation trees to, for example, route a request for a given key to the right DHT node. Indeed, rather than export a general DHT interface, Plaxton et al.'s [13] original application makes use of hierarchical aggregation to allow nodes to locate nearby copies of objects. It seems appealing to develop a SDIMS abstraction that exposes this internal functionality in a general way so that scalable trees for aggregation can be considered a basic system building block alongside the distributed hash tables.

In [1], we make an initial case for the fusion of Astrolabe aggregation abstraction with DHTs.

1. We expose a DHT system's internal trees as an aggregation abstraction by aggregating an attribute along the tree corresponding to the hash of the attribute name. Thus each different attribute is aggregated along a different tree. This approach gives a SDIMS *scalability* with respect to both nodes and attributes.
2. We provide a flexible API that lets applications control the propagation of reads and writes and thus trade off update cost, read latency, replication, and staleness.
3. We augment DHT algorithm, Pastry [17], to ensure *path convergence* and *path locality* properties in order to achieve *autonomy* and *isolation*.
4. We provide *robustness* to node and network reconfigurations by (a) providing temporal replication through lazy reaggregation that guarantees eventual consistency and (b) ensuring that our flexible API allows demanding applications gain additional robustness by either using tunable spatial replication of data aggregates and/or performing fast on-demand reaggregation to augment the underlying lazy reaggregation.

This position paper describes the broader research agenda. In going forward, we need to further refine the above mentioned techniques to reach the goals of SDIMS.

4 Scalability

Existing DHTs can be viewed as a mesh formed of several trees. Suppose the node responsible for a key k is $root_k$. The paths from all nodes for a key k form a tree rooted at the node $root_k$ — say $DHTtree_k$. By aggregating an attribute along the tree $DHTtree_k$ for k =*hash(attribute type, attribute name)*, different attributes will be aggregated along different trees and hence the scalability. The experimental results in [1] demonstrate the scalability of this approach with both the number of nodes and the number of the attributes.

Research Issue: Composite Queries While aggregating different attributes along different trees provides scalability with respect to attributes, solving composite queries involving two or more attributes becomes hard. For example a probe like *find a nearest machine with load less than 20 percent and has more than 2 GB of memory*. If query compositions are known in advance, then attributes can be grouped and can be aggregated along one tree. For example, load and memory of machines can be aggregated along one tree if queries as in the above example are very common. But by grouping extensively, we lose the property of load balancing. This tradeoff presents a fundamental limitation of distributing attributes across trees.

Handling ad-hoc composite queries, whose compositions are not known in advance, is more complicated. Here we outline our approach in handling the queries with OR and AND operations: (1) *a* OR *b*: Walk along trees corresponding to both attributes *a* and *b*. (2) *a* AND *b*: Guess the smaller of the trees corresponding to *a* and *b*, and compute the predicate along the tree. Two approaches can be used to determine the size of the trees: (a) Along with the computation of the aggregation function for an attribute, maintain a count of the number of contributing nodes or (b) Use statistical sampling techniques – randomly choose a small percentage of nodes and evaluate the attributes. For handling general logical expressions, convert the logical expressions to their Disjunctive Normal Forms (DNF) and use above AND operation for each conjunctive term.

5 Flexibility

A SDIMS should provide *flexibility* for applications to handle (1) a broad range of read and write request ratios of the attributes, (2) spatial heterogeneity of the requests, and (3) temporal heterogeneity of the requests. As explained in Section 2, different attributes will have different read write load patterns and a single policy optimized for one read-write ratio may be inefficient for attributes with different read-write ratio.

The spatial heterogeneity of the requests for an attribute need to be considered as for some attributes, only few nodes perform read and write operations. For example in a multicast session, typically, only members post messages to the group and hence need to know about the multicast tree information.

While an attribute like *cpu-load* that is set to be updated periodically has a nice temporal homogeneity with respect to update operations, most other attributes like files, multicast membership, etc., will have a strong temporal heterogeneity in the update and probe patterns due to sudden popularity, diurnal patterns of the users, etc.,

Our research agenda is to build a clean framework for adaptation that provides several mechanisms with different consistency guarantees and enable applications to choose a policy based on their requirements.

In [1], we provide three flexible API: *install*, *update* and *probe* that enables application to control the propagation of reads and writes and thus tradeoff update communication cost, read latency, replication and staleness. The *install* API takes *up* and *down* arguments to determine how many levels up any update is forwarded and how many levels down the aggregated value is propagated respectively. And during *probe*, the query is sent up towards the root and might dispersed down the tree to gather the information depending on how far updates and aggregated values are propagated. In [1], we present simulation results illustrating the effectiveness of the flexible API in solving the first issue.

The same flexible API seems to be enough, with minor modifications, to handle the spatial and temporal heterogeneity. The spatial heterogeneity can be supported by restricting the dispersal of aggregation values down along the paths to only interested nodes. The temporal heterogeneity can also be supported in the same framework through supporting dynamic adaptation of *up* and *down* values.

Several research issues arise in supporting dynamic adaptation to temporal heterogeneity of operations: (1) What should be the monitoring interval?, (2) What values to adapt to?, (3) How long should a lease be?, (4) How to invalidate the leases?, and (5) How old a value can be used in responding to a probe? We exploit the existing extensive research in distributed file systems(e.g., [9]) and web caching(e.g., [7]) to answer these issues.

6 Robustness

Reconfigurations are norm in a distributed system. The straightforward approach is to reaggregate the data on failures. But reaggregation affects availability and has high costs [2]. Hence the two goals that we try to achieve are: (i) to reduce the number of reconfigurations to avoid costly reconfigurations and (ii) to mask the unavailability during reconfigurations to avoid the probe response delays. We propose two approaches – K-way hashing and Supernodes [11], compare them analytically, and discuss the pros and cons.

Reconfigurations are costly A SDIMS exposes DHT trees for aggregation. Any change in the structure directly effects the aggregated values in the system and hence the need for the reaggregation and the requirement to reduce the frequency of reconfigurations. Suppose m attributes are configured to be update-UP and N be the number of nodes in the system. Assuming a well balanced DHT, the amount of information exchanged between nodes when a node i fails is $\leq \frac{m}{2}(\log N - 1) + \sum_{l=1}^{\log N - 1} \frac{m}{2^{l+1}}(\log N - (l+1))c_l^i$, where c_l^i denotes the children at or below level l of the node i (Refer to Figure 1). This is $O(m.\log^2 N)$ communication cost assuming a constant number of children at each level for a node. A new node joining the network also inflicts similar amount of communication cost.

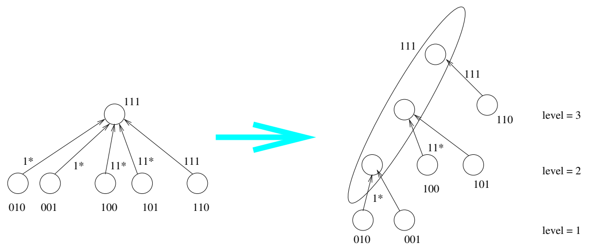

Fig. 1: Children of a node arranged in levels

Unavailability during reconfigurations The reconfigurations in a SDIMS are handled in three stages: (i) **Detection** DHT detects the reconfiguration and informs the SDIMS layer, (ii) **Structure repair** The DHT performs the repair algorithms, and (iii) **Reaggregation** the SDIMS layer reaggregates the data. The probes during the reconfigurations may be delayed or may get stale values or both.

Replication in Space The updates can be propagated to all nodes, aka full replication, to counter the unavailability during reconfigurations. Flexible API in our current system allows the replication of aggregated value on all nodes so that queries perceive no delays while guaranteeing eventual consistency. But such full replication is inefficient as the updates have to be propagated to all nodes. The flexible API also allows the replication to only a few nodes. But this does not reduce the cost of reconfigurations as the reaggregation has to be done as soon as possible to avoid staleness of the responses.

Here we explore two other approaches that does controlled replication in space that reduce the costly reaggregations while masking the unavailability: (1) K-way hashing and (2) Supernodes. Both approaches achieve the required robustness goals at the increased cost of communication during the normal operation.

6.1 K-way Hashing

In k-way hashing, each attribute is aggregated along k different trees. These trees are computed using prechosen k hash functions. Updates and probes are sent along all of the k trees for an attribute. This approach masks unavailability at all stages of the reconstruction in the face of reconfigurations.

6.2 Supernodes

The key idea is that a single NodeId is shared by multiple nodes. All nodes that are part of a supernode replicate the behavior of a single node in normal DHT. Two approaches are possible in forming supernodes: (1) Shadowing: Each node selects k shadow nodes to form a supernode with NodeId equal to the node's nodeId and (2) Clustering: Few nodes group together to form a supernode.

Shadowing: In Shadowing technique, the number of supernodes in the system are same as the number of nodes. As long as one of the shadows of a node is up, the node's failure and recovery does not effect the system. If a shadow of a node fails, the node creates a new shadow. Based on the information that is shadowed, we have two variants: (A) shadow only DHT connectivity information: Since the structure is maintained, queries can be answered by aggregating the data and (B) shadow both connectivity information and aggregation data: this is full behavioral replication at both DHT layer and the SDIMS layer and hence applications does not perceive any difference during a node failure. While approach A masks DHT repair time, the data still needs to be reaggregated. Approach B masks both unavailability and reduce reconfiguration costs but at the expense of extra communication cost for data shadowing during normal operation.

Clustering In Clustering technique, few nodes cluster together to form a supernode. When the number of nodes increases a *high threshold* mark, the cluster is split into two smaller clusters. Similarly, when the number decreases below a *low threshold* mark, the cluster is merged with some other cluster. The *high threshold* mark should be greater than twice the *low threshold* mark for splitting to be possible and should be much more to avoid frequent splits and merges.

Issues in supernodes (i) How many replicas to choose? A larger k implies stronger robustness properties but at the cost of higher bandwidth requirements. (ii) How to choose the replicas? Choosing nearby nodes reduces the communication costs while making the scheme vulnerable to domain failures. (iii) How to perform reconstruction transparently? During reconstructions both old and new supernodes has to be maintained. (iv) How to maintain consistency between the replicas? Gossiping between replicas guarantee eventual consistency.

6.3 Analytic comparison

Assume p be the probability of a node failure and l be the path length from a node to the root for an attribute. The probability of an access failure at the node in case of k-way hashing, P_{hash}, is (one path fails)^k = $(1-(1-p)^l)^k$ and in case of k-way shadowing, P_{shadow}, is $1-$(atleast one node does not fail)^l $= 1-(1-p^k)^l$ For k-way clustering, the probability of an access failure is similar to P_{shadow} with l replaced by $l - \lg k$ in the exponent.

These functions are compared in Figure 2 for $l = 10$ and $k = 4$. Clearly, supernode approach is more robust than K-way hashing.

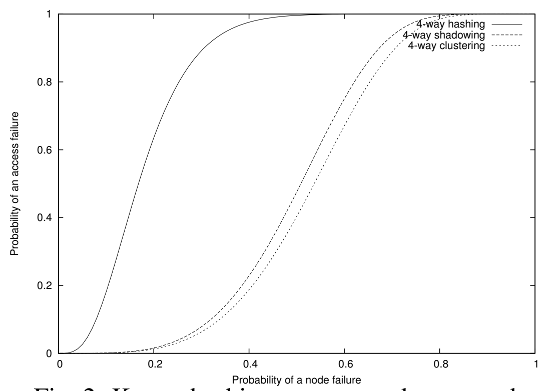

Fig. 2: K-way hashing vs supernode approach

6.4 Discussion

While K-way hashing is simpler than supernodes, the probability of an access failure is greater than in supernode approach for same values of replication. The DHT maintenance overheads in K-way hashing during normal operation are none but reconstruction of the DHT is needed during reconfigurations. In Supernodes, reconstructions can be delayed and hence no overhead during reconfigurations but requires more overhead for normal DHT maintenance as each node needs to keep track about the connectivity information of k other nodes.

Clustering vs. shadowing: Shadowing has the advantage that a node has the flexibility to choose any other node as its shadow. enable us to handle heterogeneity : choose more stable node as a shadow, choose more capable node as a shadow, etc.,

Replication in time With small factor of replication in space, in cases where all replicas are down, reaggregation is necessary. In [1], we provide two-fold replication in time mechanisms: (i) lazy reaggregation, and (ii) on-demand aggregation. During lazy reaggregation, probes might get stale answer. But applications can force on-demand reaggregation at the cost of increased response time and communication cost. Lazy reaggregation limits the bandwidth used for reconfiguration and hence controls the cost of reconfiguration. Note that these approaches does not mask unavailability during failure detection and DHT repair stages.

7 Autonomy and Isolation

In [1], we present modifications to Pastry [17] that guarantee autonomy and isolation requirements. By maintaining separate leafsets for each sub-domain in which a node is a member, a simple modification to routing protocol ensures that there is single node in each domain to which all requests for an attribute are routed. The experimental results in [1] show that the increase in path length due to these modifications is very minimal.

8 Conclusions

This paper makes a case for developing a general Scalable Distributed Information Management System (SDIMS) abstraction that *aggregates* information about large-scale networked systems and that can serve as a basic building block for a broad range of large-scale distributed applications. It also outlines the requirements of a SDIMS, describe an initial prototype, lays out key research challenges, and provides directions for tackling those issues.

References

[1] Anonymous. SDIMS: A Scalable Distributed Information Management System.

[2] M. Baker. *Fast Crash Recovery in Distributed File Systems*. PhD thesis, University of California, Berkeley, CA 94720, Jan. 1994. Technical Report UCB/CSD 94/787.

[3] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-bandwidth Multicast in a Cooperative Environment. In *Proceedings of the 19th ACM SOSP*, October 2003.

[4] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. SCRIBE: A Large-scale and Decentralised Application-level Multicast Infrastructure. *IEEE Journal on Selected Areas in Communications (JSAC) (Special issue on Network Support for Multicast Communications)*, 2002.

[5] J. Challenger, P. Dantzig, and A. Iyengar. A scalable and highly available system for serving dynamic data at frequently accessed web sites. In *In Proceedings of ACM/IEEE, Supercomputing '98 (SC98)*, Nov. 1998.

[6] R. Cox, A. Muthitacharoen, and R. T. Morris. Serving DNS using a Peer-to-Peer Lookup Service. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, March 2002.

[7] Z. Fei. "a novel approach to managing consistency in content distribution networks". In *Proceedings of Web Caching and Content Distribution Workshop*, 2001.

[8] Ganglia: Distributed Monitoring and Execution System. `http://ganglia.sourceforge.net`.

[9] C. Gray and D. Cheriton. Leases: an efficient fault-tolerant mechanism for distributed file cache consistency. In *Proceedings of the twelfth ACM symposium on Operating systems principles*, pages 202–210. ACM Press, 1989.

[10] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks. In *Proceedings of the OSDI*, December 2002.

[11] D. Malkhi. Dynamic Lookup Networks. In *Proceedings of the International Workshop on Future Directions in Distributed Computing (FuDiCo)*, 2002.

[12] P. Maymounkov and D. Mazieres. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. In *Proceesings of the IPTPS*, March 2002.

[13] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In *ACM Symposium on Parallel Algorithms and Architectures*, 1997.

[14] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content Addressable Network. In *Proceedings of ACM SIGCOMM*, 2001.

[15] R. V. Renesse, K. P. Birman, and W. Vogels. Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining. *ACM Transactions on Computer Systems*, 21(2):164–206, May 2003.

[16] T. Roscoe, R. Mortier, P. Jardetzky, and S. Hand. InfoSpect: Using a Logic Language for System Health Monitoring in Distributed Systems. In *Proceedings of the SIGOPS European Workshop*, 2002.

[17] A. Rowstron and P. Druschel. Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-peer Systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms(Middleware)*, November 2001.

[18] S.Ratnasamy, M.Handley, R.Karp, and S.Shenker. Application-level Multicast using Content-addressable Networks. In *Proceedings of the NGC*, November 2001.

[19] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

[20] S.Zhuang, B.Zhao, A.Joseph, R.Katz, and J.Kubiatowicz. Bayeux: An Architecture for Scalable and Fault-tolerant Wide-Area Data Dissemination. In *Proceedings of the NOSSDAV*, June 2001.

[21] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical Report UCB/CSD-01-1141, UC Berkeley, Apr. 2001.