CAN A DECENTRALIZED STRUCTURED STORAGE SYSTEM SUCH AS CASSANDRA PROVIDE AN EFFECTIVE MEANS OF SPEEDING UP WEB ACCESS TIMES

Dennis Guster
Department of Information Systems
Saint Cloud State University
St. Cloud, MN 56301
dcguster@stcloudstate.edu

Andres Q. O'Brien
Department of Information Systems
Saint Cloud State University
St. Cloud, MN 56301
oban0601@stcloudstate.edu

Laura Lebentritt
Computer Management Information Systems
University of Maryland University College
Adelphi, MD 20783
lauraleigh4097@att.net

ABSTRACT

The growth of the Internet and volume of data that it transports have increased the need for effective database models. One of these models is Cassandra, which offers both cost effectiveness and features ease of use, high scalability, and can run on inexpensive hardware. Because mechanical hard drives still dominate and this is the bottleneck in the data retrieval process, it is reasonable to investigate optimization by storing data on multiple disks, distributed across multiple devices. This methodology suggests a reduction of data access time can be achieved using Cassandra. Therefore, research is needed to delineate the advantages of distributed databases and this study addresses this with data obtained from a basic configuration of a Cassandra database. Data collected from the Cassandra test bed revealed that in general, a distributed database using additional nodes could reduce latency. However, diminishing returns were observed as additional nodes were added into the experiment.

INTRODUCTION

In today’s global oriented business world the vast majority of applications are delivered via the web. This global strategy has often necessitated the support of huge pools of users, which need to extract data on both a read and write level from a database structure. If one looks at this issue historically, before applications became web applications (pre-http), the maximum number of potential users was typically limited to the size of the connected private network, which often translated to a maximum value in the thousands of users. However, the transition to Internet based web applications has increased the pool of potential users to millions. Supporting this huge pool is not trivial and requires a well thought out strategy in regard to reliability, performance and scalability. Therefore, it is crucial to optimize the stored data, which will need to be extracted, processed and forwarded to a web client. Although there are faster alternatives (such as solid state drives) the traditional mechanical hard drive technology still dominates the landscape and is the slowest part of the information retrieval process. This mechanical structure often means that even with state of the art disk drive technology, adequate performance cannot be obtained with the most intensive web applications involving the “millions of hits scenario.”

Because this “millions-to-one” mechanical bottleneck is so prevalent in enterprise computing it makes sense to investigate methods to optimize large scale storage. Further, because mechanical storage technology is still so prevalent it seems logical to investigate methods that improve upon its characteristics. Prior research has indicated that using multiple nodes and creating a distributed database can in fact be quite effective (Elnikety et al. [5],Kanitkar [11],Kanitkar and Delis [12], Brown et al. [3]). While this basic method offers a solution, often it involved extensive setup and required considerable personnel time making its cost-effectiveness questionable. The recent availability of shareware products that take advantage of existing nodes in the enterprise make the harnessing of distributed database logic more practical and cost effective. One of the most popular is a product called Cassandra (Lakshman and Malik [13]). Lakshman and Malik describe Cassandra as follows. Cassandra can be viewed as a distributed storage system capable of managing huge amounts of structured data distributed across many commodity servers, while still providing highly available service and fault tolerance. Cassandra is designed to operate in an infrastructure containing hundreds of nodes (often distributed across multiple data centers). When operating at this scale, it is not unusual for components to fail continuously. Cassandra is designed to manage these drive failures ensuring the reliability and scalability of the software systems that rely on this service. While Cassandra does resembles a database (DB) because it is based on several DB design and implementation strategies, Cassandra does not support a full relational data model; instead, it provides clients with a simple data model that supports dynamic control over data layout and format.

Therefore, the purpose of this paper is to investigate the effectiveness of using the Cassandra model in regard to performance gains and ease of use. A series of experimental trials will be launched within a series of existing nodes within the author’s autonomous system under varying loads to ascertain the performance gain that might be realized by using the Cassandra system.

REVIEW OF LITERATURE

Background

The concept of distributed databases has been around for quite some time (Abadi, D. J., [1]). During its inception the potential end-users were limited to fairly large organizations with sophisticated IT staffs. However, distributed database technology has matured in recent years to become more-user friendly, cost-effective and easily maintained by most organizations of any size. In terms of benefits, distributed databases often go beyond just the desired improvement in access time reduction. Ozsu and Valduriez [15], report that the benefits might also include: ease of management of distributed/replicated data, improved reliability through distributed transactions, better performance, and easier, more economical system expansion.

Data is at the heart of most businesses and storing/accessing that data efficiently is crucial to the business's success. Recent research has delineated the advantages of virtualization in the management of large numbers of interrelated applications (Hemminger et al. [9], Safonov et al. [16], Guster et al.[8]). Therefore, the concept of virtualization is also important in the design of effective distributed databases. Specifically, Xu et al. [17] address this issue and have developed a framework/implementation plan for a virtual database management (VDM) system which can integrate different types of data resources. Further, this system allows the user of virtual database space to manage distributed resources using SQL (structured query language) type logic. This VDM system is especially well suited to many scientific research areas such as the spatial information grids and data integration among different organizations.

The characteristics of the data typically influence how it is stored and indexed. However, there is a degree of commonality in regard to how it might be accessed. Given the world-wide dependence on the internet, web based delivery has certainly become the dominant model. With that model there is an expected latency on the part of end-users (typically 3 seconds or less). This metric is not always easy to achieve and becomes more elusive when inquiries to the database are lengthy. So therefore, the efficiency in which the data is stored and staged through queuing becomes an integral component in meeting the three second delay target. The work of Zimmermann et al [18] addresses this question and delineates how the queuing structure could be optimized for a web-based GIS application.

Security Concerns

While the benefits of using a distributed architecture in regard to performance, reliability and ease of management are apparent, the architecture does create potential security concerns. These concerns often stem from the fact that the database is not located entirely on one physical host, but rather, the data is distributed across hundreds of hosts (both physical and virtual). One has to assume that if any one of the hosts is corrupted then the validity of the entire data may become suspect. However, in the distributed world there are multiple copies of the primary data stored within the grid or cloud. So therefore it is logical to remove a host that has been attacked from the grid and then access that data from another host that has a replica of the data in question. This methodology has been successfully employed by Iancu and Ignat, [10]. Their work also stresses the importance of devising sound trust relationships within the grid to minimize the likelihood that attacks can be successful in the first place. This logic necessitates an integrated access control model that carefully evaluates trust relationships. For example, if the user generating a particular transaction has root level privileges on host "A" and needs to extract data on host "B" (host "A" is trusted by host "B") it doesn't necessarily follow that the root level privileges should follow the user to host "B". Liu, Han and Shen [14], have addressed this problem and devised a formal distributed database security definition. Further, they have successfully implemented this security strategy for a relational Database integrated access control model (DBIAC) based on APT (Authorization Propagation Tree).

Architecture and Tradeoffs

The best architecture to facilitate an effective distributed database system is still evolving. While a limited peer to peer (P2P) architecture has been a popular/effective choice and is widely used on a cluster level, there are other options. Further, the design of P2P is still evolving and current designs have limitations. Bonifati et al. [2], provide analysis in regard to comparing P2P with DB-centric models and have devised a taxonomy of the most prominent research efforts towards the realization of full-fledged P2P databases. Optimizing this underlying infrastructure is critical in maximizing the effectiveness of a distributed database. However, there are often tradeoffs in this process which are described by the CAP concept. CAP basically states that in building a DDBS, designers can choose two of three desirable properties: consistency (C), availability (A), and partition tolerance (P) Abadi, [1].

Abadi, [1] further explains the need to look beyond the CAP model: A more complete portrayal of the space of potential consistency tradeoffs for DDBSs can be achieved by rewriting CAP as PACELC (pronounced "pass-elk"). In this model, if there is a partition (P), how does the system tradeoff availability and consistency (A and C); else (E), when the system is running normally in the absence of partitions, how does the system trade off latency (L) and consistency (C)? Note that the latency/consistency tradeoff (ELC) only applies to systems that replicate data. It is clear that the tradeoffs involved in building distributed database systems are complex, and neither CAP nor PACELC can explain them all. However, the incorporation of consistency/latency tradeoffs within a modern DDBS design certainly is important enough to warrant analyzing the tradeoffs via the models presented in architectural discussions.

Cassandra

The Apache Cassandra project has provided a new and inexpensive option to support applications that require a high performance distributed database. The Apache Cassandra database is billed as the right choice when scalability and high availability are desired. Cassandra accomplishes this without compromising performance. It provides linear scalability and proven fault-tolerance while using commodity hardware or a cloud infrastructure which makes it the perfect platform to support mission-critical data. Further, Cassandra's support for replicating across multiple data centers is often viewed as the best-in-class which provides lower latency to your end-users giving them the confidence to survive regional outages (cassandra.apache.org) [4]. Specifically, Cassandra is a distributed key-value store style database with the ability of scaling to very large data sets with no single failure point. The underlying hardware layer can be a wide variety of architectures including but not limited to server nodes, racks, and several data centers. The growing need to store and access data, which is rapidly reaching the petabyte scale, has necessitated the need for distributed databases such as Cassandra which are capable of operating at that massive level using existing technology. Cloud computing has changed the scale potential of autonomous systems. In the past, while hosts with RAIDS for the most part were reliable, there still was the issue that the failure of that host would jeopardize functionality. Cloud computing utilizes distributed processing at its best and uses replication across hosts to ensure fault tolerance and high-performance access to the data. However, Cassandra also offers some flexibility in regard to how it is designed to deal with component failure. Specifically, Cassandra can be configured to promote high system availability by compromising data consistency, but will allow the user to select the degree of this tradeoff. In other words, data stored within the Cassandra structure can be configured to replicate across N different peers within its host cloud while employing a gossip protocol to ensure each node will maintain its state in relation to its peers (Featherston, 2010) [6].

METHODOLOGY

An existing, but outdated computer cluster within the authors' autonomous system was allocated to the project because each node had four 250GB mechanical hard drives. A test bed was setup so that Cassandra could be implemented to a maximum of 16 nodes. Each node had two 64bit 2Ghz processors and 2GB of memory. All nodes in the cluster were connected via a 1Gbs Ethernet connection. Two metrics were chosen to determine the efficiency of using the distributed database. These metrics were: Average latency, which is a measure of how long any given transaction might have to wait to be processed and elapsed time, which is a measure of how long it will take to complete either the read or inset session. Sessions of varying intensities were used to determine how Cassandra might scale as the workload and number of nodes used was increased. The sessions varied from a minimum of 5,000 records to a maximum of 1million records. The initial tests depicted below were generated by a single client and rpc_max_threads was set at 50 while the maximum number of concurrent reads/writes was set at the default value of 32. Further, the replication factor was set to 1 so the data during inserts was actually inserted in the primary partition as well as one replication partition. This would be considered a minimum baseline configuration for a distributed database.

RESULTS

The results are reported below in Table 1. These results indicate that adding additional hosts did not always result in an improvement in average latency or a reduction in elapsed time. For example, in the case of the 5,000 record intensity trial the best latency was achieved at 8 hosts on the insert level. Even at 1 million records 8 hosts still provided the smallest latency. However, at times it does appear that adding additional hosts does have a marked effect on latency. This is particularly true at the 1 million insert level whereby the latency was reduced from .0023 to .0011 by running it on 8 rather than just 2 hosts. To a degree it is confounding because at certain points the latency increases when additional hosts are added. Once again at the 1 million insert level the latency increases by .0002 when going from 8 to 16 hosts. This is logical if one remembers that there is additional communication overhead as hosts are added to the mix. So often it is a matter of finding the ideal number of hosts for any given load level. For inserts it varies with the load: 2 hosts for 5,000 records, 8 hosts for 10,000 records, 16 hosts for 50,000 records, more than 2 hosts for 100,000 records and 8 hosts for 1 million records. The same logic pretty much holds true for reads as well.

	5K records		*10K records*		*50K records*		*100K records*		*1000k records*	
# of Hosts	**READ**	**INSERT**	**READ**	**INSERT**	**READ**	**INSERT**	**READ**	**INSERT**	**READ**	**INSERT**
(2 hosts)	0.0045,1	0.0102,3	0.0039,2	0.0342,8	0.0012,8	0.0020,13	0.0031,11	0.0026,16	0.0037,97	0.0023,106
(4 hosts)	0.0053,1	0.0047,3	0.0013,2	0.0189,6	0.0014,8	0.0060,19	0.0022,10	0.0012,20	0.0015,70	0.0018,109
(8 hosts)	0.0038,1	0.0069,4	0.0018,2	0.0042,4	0.0015,7	0.0018,17	0.0008,10	0.0012,16	0.0008,71	0.0011,100
(16 hosts)	0.0020,1	0.0077,4	0.0027,2	0.0294,7	0.0014,9	0.0013,13	0.0010,14	0.0012,18	0.0011,71	0.0013,104

Table 1: Average Latencies/Elapsed Times for a Cassandra DB at Various Record Intensities (**avg_latency,elapsed_time).**

In terms of elapsed time the value observed is definitely a function of the intensity. In other words, one would expect that it would take longer to process 1 million records than it does 5,000. In a linear world then one could expect the elapsed time to be 200 times longer (1,000,000/5,000). However, there appears to be some degree of efficiency when dealing with large record sets within the Cassandra logic, because at the most efficient level when inserting 1 million records, the elapsed time is only about 25 times greater than at 5,000 records. However, adding additional hosts has only a minimal effect in this endeavor. When inserting 1 million records 8 hosts were only 6 seconds quicker than 2 hosts.

DISCUSSION AND CONCLUSIONS

From the data contained in Table 1 it is clear that additional hosts can reduce both the latency and the elapsed time. At first, the magnitude achieved might seem somewhat suspect. However, when one considers the type of applications that might be supported then the value is much clearer. For example, if the application is e-commerce then response time is critical, a total end-to-end response time of 3 seconds or less is critical to meet a customer's expectations (actually the quicker the better). If this target metric is not met customers may get impatient and migrate to another site, perhaps never to return. Therefore, a profit making company would be well advised to evaluate the performance gains Cassandra could provide.

Given the data above on the read level at 5,000 records being able to drop the latency from .0045 to .0020 or at 1 million records being able to drop the latency from .0037 to .0011 could be significant in meeting the 3 second target. While ~.0025 of a second may not seem that significant, one needs to look at the total response time model which is quite complex and involves a number of components.

A good representative example of this model is offered by Flemming [7]: **User - Application-Command _CPULocalComputer _NICLocalComputer _Network-Propagation _Switch _Network-Propagation _Switch _Network-Propagation _NICFile-Server _CPUFile-Server _SCSI-bus _DiskRead**, (_…then traverse path in reverse for the reply).

Given that the network delay on the Internet in the US might often take .5 seconds in each direction it is important to optimize each of the parameters. Further, one needs to realize that this whole algorithm is based on queuing theory, which means that there is an interaction among all the parameters. In other words, a delay of .0005 instead of .0001 at the first parameter won't simply result in .0004 seconds of additional response time. Rather, it will propagate through the entire algorithm and the delay will get a little longer with added wait time at each successive parameter. To put this in perspective, if one assumes a geometric progression through all 12 parameters in the algorithm above the result in total added delay would be close to 1 second (.8192).

Besides the potential for performance enhancement the capabilities of Cassandra to support fault tolerance are quite useful. Not only does Cassandra support the concept of replication on inexpensive hardware it allows its configuration to be tuned as well. This allows small/medium size companies to adopt Cassandra due to its cost effectiveness and ease of use. Further, it could be considered an integral component of any disaster recovery plan. It could easily be configured to replicate within a company's existing cloud or within a cloud at a remote site making it simple for a company to have copies of their mission critical data at several widely distributed remote locations.

The data contained herein, while offering some base line information does not deal with all of the complexities of implementing Cassandra. First, a single client generated the workload, which is not realistic. Second, this data was collected using a configuration that featured only one replica. Third, while a million transactions seem adequate, to obtain a representative sample for some systems would require an even larger sample size. Fourth, the number of physical nodes was limited to 16. Though this is a decent starting point, in a production world one would expect the number of nodes to far exceed this value. Therefore, future research will need to address each on these variables to ascertain the effectiveness of Cassandra in various applications and under a variety of different loads.

REFERENCES

[1] Abadi, D. J., "Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story", Computer, vol. 45, no. 2, pp. 37-42, Jan. 2012, doi:10.1109/MC.2012.33.

[2] Bonifati, Angela, Panos K. Chrysanthis, Aris M. Ouksel, Kai-Uwe Sattler, "Distributed Databases and Peer-to-Peer databases: past and present. ACM SIGMOD Record archive, Volume 37 Issue 1, March 2008 Pages 5-11 ACM New York, NY, USA doi>10.1145/1374780.1374781.

[3] Brown, C., Guster, D. C., & Krzenski, S. (2007). Can Distributed Databases Provide an Effective Means of Speeding Up Web Access Times?. *Journal of Information Technology Management*, *18*(1), 1-15.

[4] http://cassandra.apache.org/.

[5] Elnikety, S., Tracey, J., Nahum, E., and Zwaenepoel, W. "A Method for Transparent Admission Control and Request Scheduling in E-Commerce Web Sites," Proceedings of the 13th International Conference on World Wide Web, 2004, pp. 276-286.

[6] Featherston, D. (2010). Cassandra: Principles and Applications. http://disi.unitn.it/~montreso/ds/papers/Cassandra.pdf.

[7] Fleming, D. (2004). Network Response Time for Efficient Interactive Use. Proceedings of the 20th Computer Science Seminar, Addendum-T2-1. RIP, Hartford Campus, April, 24.

[8]Guster, D. C., Hemminger, C., & Krzenski, S. (2009). Using Virtualization to Reduce Data Center Infrastructure and Promote Green Computing. *International Journal of Business Research, 9*(6), 133-139.

[9]Hemminger, C., Rogers, D. C., & Guster, D. C. (2010). Planning and Managing the Data Center to Green Computing. *International Journal of Business Research, 10*(4), 105-113.

[10] Iancu, Voichija & Iosif Ignat, "A peer-to-peer consensus algorithm to enable storage reliability for a decentralized distributed database," aqtr, vol. 2, pp.1-6, 2010 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR), 2010.

[11]Kanitkar, V. “Collaborative and Real-Time Transaction Processing Techniques in Client-Server Database Architectures," Polytechnic University, Volume 61, Number 04B, 2000, pp. 2036.

[12] Kanitkar, V. and Delis, A. “Distributed Query Processing on the Grid," IEEE Transactions on Computers, Volume 51, Number 3, 2002, pp.269-278.

[13] Lakshman, A. and Malik, P. “Cassandra: A Decentralized Structured Storage System," ACM SIGOPS Operating systems Review, Volume 44, Numbers 2, 2010,pp. 35-40.

[14] Liu, Xin, Zhen Han, Chang-xiang Shen, "An Integrated Access Control Model of Distributed Database Systems," icicic, vol. 3, pp.209-212, First International Conference on Innovative Computing, Information and Control - Volume III (ICICIC'06), 2006.

[15] Özsu, M. T., & Valduriez, P. "Distributed Database Systems: Where Are We Now?," Computer, vol. 24, no. 8, pp. 68-78, Aug. 1991, doi:10.1109/2.84879.

[16]Safonov, P. I., Guster, D. C., & Hemminger, C. (2011). Employing host virtualization and symmetric multi-processing as a strategy for improving performance in computentionally intense problems. *Issues in Information Systems, XII*(1), 357-365.

[17] Xu, Wenhao, Jing Li, Yongwei Wu, Xiaomeng Huang, Guangwen Yang, "VDM: Virtual Database Management for Distributed Databases and File Systems," gcc, pp.309-315, 2008 Seventh International Conference on Grid and Cooperative Computing, 2008.

[18] Zimmermann, Roger, Wei-Shinn Ku, Wei-Cheng Chu, “Efficient query routing in distributed spatial databases”. GIS '04 Proceedings of the 12th annual ACM international workshop on Geographic information systems Pages 176 - 183 ACM New York, NY, USA ©2004 table of contents ISBN:1-58113-979-9 doi>10.1145/1032222.1032249