# AN EMPIRICAL ASSESSMENT OF PERFORMANCE AND SCALABILITY OF DECENTRALIZED DISK STORAGE FOR REAL-TIME BUSINESS APPLICATIONS

Dennis C. Guster, Saint Cloud State University
dcguster@stcloudstate.edu

Mark B. Schmidt, Saint Cloud State University
mbschmidt@stcloudstate.edu

Erich P. Rice, Saint Cloud State University
rier1201@stcloudstate.edu

## ABSTRACT

The growth of the Internet has increased the need for effective databases. One is Cassandra, an open source database by the Apache Software Foundation, which claims high scalability and is able to run on commodity hardware. Because mechanical hard drives are 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 by using Cassandra. Research is needed to determine advantages that can be obtained by using distributed databases. This study obtained data from a basic configuration of Cassandra, and the test bed revealed that a distributed database using additional nodes could reduce latency and add efficiency. However, as more nodes were added to Cassandra, diminishing returns were observed and the addition of nodes added only slightly to the efficiency of the database.

## INTRODUCTION

The Internet has revolutionized the basic business model in which customers need to be physically near the product they wish to purchase, or be on a mailing list for a catalog that then allows them access to it through the mail. Its worldwide connectivity has for many businesses expanded their customer base from maybe 10,000 people residing in their local region to millions of people around the world who have web access. While this new global commerce model is most gratifying from a profit perspective to the individual companies, it necessitates the support of huge pools of e-commerce users, who need to extract data on both a read and write level from a backend database structure.

To put the ramifications of this massive growth in perspective, a quick review of the hardware architecture needed to support it is in order. Before the rapid worldwide growth of the Internet,

the maximum number of potential users of a system was typically limited to the size of the connected private network that supported it; this typically was limited to perhaps a few thousand users. However, the new global Internet based e-commerce web model has resulted in user groups in the millions. Supporting a user pool of this magnitude is not trivial and opens a question in regard to the functionality of the system related to reliability, performance, security and scalability. Further, the training level of personnel required to run such a system is much greater than in just a regional or private network model. The primary metric to be evaluated in this paper is performance or more specifically the number of read or insert operations performed in a given amount of time by the database system. To help put those metrics in perspective a brief discussion concerning response time is warranted. Because the primary delivery mechanism for business on the Internet is e-commerce, which is usually web based, a target latency of about three seconds or less is often the goal (Brown, Guster, & Krzenski, 2007). Achieving this metric can be challenging and can become more elusive as the size of the database increases, as well as when the volume and complexity of inquiries increase. Therefore, the design of the architecture used to store the data and the staging (queuing) process when extracting that data becomes an integral component in meeting the three second delay target. For a more detailed discussion see Zimmermann, Wei-Shinn, and Wei-Cheng (2004) that addresses this question and delineates how the queuing structure could be optimized in a web-based GIS application.

Given the available hardware then what is the most efficient and cost-effective way to optimize the stored data, which will eventually be extracted, processed and then forwarded to a web client? While faster alternatives exist (like solid state drives) the traditional mechanical hard drive technology is still the primary architecture for large scale data storage. Because these drives are mechanical in nature they are typically the slowest part of the information retrieval process. In the traditional sense, involving a few large mechanical drives, the target performance goal of "three seconds or less" cannot be achieved in intensive web applications that involve an end user pool which generates a "millions-of-hits" restriction point, the work by Otey, Parthasarathy, Wang, Veloso, and Meira (2004) looked at ways around this bottleneck.

Any business that has been able to effectively use e-commerce and thereby expand their customer base worldwide, must address this "millions-of-hits" mechanical bottleneck. It therefore makes sense to investigate methods that optimize large-scale storage. Further, because of its widespread usage and thus cost effectiveness, mechanical storage technology is still the dominant architecture and it seems logical to investigate methods that improve upon its characteristics. There are numerous studies that indicate using multiple nodes and creating a distributed database can be an effective solution (Brown et al., 2007; Elnikety, Tracey, Nahum, & Zwaenepoel 2004; Kanitkar, 2000; Kanitkar & Delis, 2002). Historically, while the basic methodology offered a solution, it often involved extensive setup and required a considerable amount of time from high-level personnel, which limited its cost-effectiveness. The widespread use of the LINUX operating system, which in addition to enhancing performance, security, and scalability created an excellent platform to deploy shareware, has created greater efficiency possibilities. There have been several shareware products that can easily reside on top of existing LINUX nodes (often in a cloud architecture), which make the harnessing of distributed database logic more practical, less personnel intensive, and hence more cost effective.

Perhaps the most popular of these shareware database products is Cassandra. Lakshman and Malik (2010) described Cassandra as a distributed storage system capable of managing huge amounts of structured data distributed across many commodity servers, and providing a high availability of service and fault tolerance. Cassandra was designed to operate in a way so it could be used on hundreds of nodes and could be distributed across multiple data centers, or Clouds (Cockcraft, 2011). When operating on this scale, it is not unusual for hardware components to fail almost continuously. Cassandra is designed to manage these drive failures ensuring the reliability and scalability of the software systems that rely on this service, in the case of Netflix, it chose a Cloud driven solution through the Amazon Web Service (AWS) (Cockraft, 2011). Although Cassandra resembles a relational database in that it incorporates database design and implementation strategies, Cassandra is not a relational database. It instead provides a simple data model that supports dynamic control over data layout and format, while providing increased redundancy and greater fault tolerance (Cockcraft, 2011).

Therefore, the purpose of this paper is to investigate the appropriateness and effectiveness of using the Cassandra model to help alleviate the "millions-of-hits" scenario, which might be encountered by a business using e-commerce on a global scale. Specifically, the system will be evaluated in regard to performance gains and the ease with which it can be implemented. To collect meaningful data, a series of experimental trials were carried out on a test bed of existing nodes within an autonomous system set up as a Cloud infrastructure, the tests were carried out under differing loads to help ascertain the performance gains that might be realized by using the Cassandra distributed database system.

## BACKGROUND AND REVIEW OF CURRENT PRACTICE

The idea behind a distributed database system is certainly not a new or novel one. The idea and implementation of a distributed database system can be traced back at least to the late 1970s and the work of Bernstein, Rothnie, Goodman, & Papadimitriou (1978). As the amount of data produced and the need to extract and manipulate it with ever increasing speed and accuracy has increased, new and more efficient ways have been developed to deal with this ever-increasing cascade of data. As Abadi (2012) pointed out, though the initial wave of potential end users were limited to large organizations such as multinational corporations and the Federal government, the newer distributed database technologies have allowed a greater number and variety of organizations to utilize them. The benefits that even small or medium sized firms can now realize from a distributed database environment can far outweigh the costs of creating and managing the system. As Oszu and Valduriez (1991) pointed out, the benefits that such a system could provide are better system performance, ease of managing replicated data, improved reliability of transactions, and hopefully an easier and more economical method of scalability.

In today's data driven world, nothing is more important to an organization than the accumulation and preservation of useful business relevant data. The storing and easy access of that pertinent data is often times of critical importance to the continued operation of a going concern. Among

the important developments in accessing and utilizing applications has been the rise of virtualization, through tools such as VMware or Microsoft's Hyper-V, and it has been noted that virtualization has created greater efficiencies of operation (Hemminger, Rogers, & Guster 2010; Safonov, Guster, & Hemminger 2011). Thus, virtualization also has a place within the distributed database realm as well. In Oracle for example, the ability to utilize a virtual private database or VPD, allows the database administrator a level of finer granularity control over both row and column level access (Afyouni, 2006). Within the distributed database realm, various ideas have been developed to create a framework around which the virtualized database space can be managed, unlike Oracle in which an individual instance or "view" of a portion of the database is given to a user. For instance, Xu, Jing, Yongwei, Xiaomeng, & Guangwen (2008) presented a methodology for a virtual database management system or VDM, which would allow the organization to integrate different data resource types and varying data sets into a cohesive unit.

The VDM architecture could also allow users of the virtual database environment to utilize a standardized method such as structured query language or SQL to manage and query the database structure. Xu et al. (2008) also pointed out that the VDM environment might be best suited to scientific areas of research or data collection, such as data integration across organizations or spatial information grids which require a high degree of cross referencing. Although the data type can cause varying differences in how it is stored and indexed within the database, due to the Internet, the typical delivery method for the indexed data is through a web interface of some kind. Thus, the earlier discussed "three second rule" comes into play and queuing and storage of the data becomes an important piece to how the system operates, and whether it meets end user expectations. Zimmerman et al. (2004) discussed how a web based Geographic Information System (GIS) could be structured that best optimizes the end user experience, by queuing the data within the distributed database structure.

Although distributed database systems have been shown to provide a great many benefits to web or cloud based architectures (Cockcraft, 2011), the best way to facilitate that architecture is still under consideration. A peer-to-peer architecture works well on a cluster level, and although there are other options, Bonifati, Panos, Aris, &Kai-Uwe (2008) provide an excellent analysis of how to best integrate a peer-to-peer architecture with a database centric model. Only through this type of effort can the true benefits of a distributed database system be realized. Other research has focused on the CAP or consistency, availability, and partition tolerance concept. Abadi (2012) points out that within the CAP methodology there are tradeoffs, and that in designing a distributed database system really only two of the three desired criteria can be reconciled.

Abadi (2012) also explains that it is important to look beyond the CAP model, which he views as too simplistic, and instead define it as PACELC. The acronym stands for (P)artition, how the system trades off between (A)vailability and (C)onsistency; (E)lse, when the system is running under normal conditions without partitions, how does the system tradeoff between (L)atency and (C)onsistency? The ELC portion of the methodology only applies to those systems that replicate

data, such as a replication (REPL) database. Nevertheless, these types of balancing decisions are needed when determining the best option for an organization's distributed database structure.

This brings us to Cassandra, which the Apache Software Foundation developed as an open source option for high-level distributed database applications (Apache, 2009). The Cassandra distributed database platform claims to provide a high level of system performance, coupled with the ability to scale up the implementation by adding additional nodes to the distributed database network. It can be implemented through either a hardware cluster, or through the means of a cloud infrastructure as was tested in Cockcraft (2011). Cassandra's ability to replicate itself across datacenters and across the cloud has lead many large companies to utilize it for their distributed database needs, as was shown in the Netflix case example (Cockcraft, 2011). The amount of data being created and stored by large Internet firms such as Netflix has grown into the petabyte range, or 10 to the $15^{th}$ power bytes of information, and ways to scale up the storage apparatus of Cassandra through the continued addition of nodes is a main selling point to its use. Cockcraft (2011) points out that, at least in the Netflix example, the increase in performance was an almost linear increase in performance as new nodes were brought online.

Cassandra can be configured also to promote a higher level of system availability, though it compromises data consistency, it will however allow the user to select the degree of this tradeoff. Thus, data stored within the Cassandra structure can be configured to replicate across N different peers within its host cloud or cluster, while employing a gossip protocol to ensure each node will maintain its state in relation to its peers (Featherston, 2010). For these reasons many have chosen to utilize the Cassandra distributed database system for their own implementations, and those of their clients (DataStax, 2013).

## METHODOLOGY

The test bed for the experimental portion of the paper was configured within an existing cloud infrastructure located within the authors' lab. From a hardware perspective each node was virtualized on a Dual Core CPU at 2.2GHz (AMD Opteron 6174) with 32GB of RAM and 2.6TB of hard drive space using Cronos VMware NFS. Each node was replicated only once, which would be considered the minimum if fault tolerance was desired as it would be in a real world enterprise level setting. The basic processing stack was set to 50 threads which is a reasonable starting point to assess the effectiveness of multiprocessing. The block of data to be inserted (written) or read consisted of five columns of data in a table and each column was 34 bytes in length. This was designed to mimic a basic inquiry/response transaction processing system.

The metrics chosen to determine the efficiency of using the distributed database were primarily based on latency. The basic metric was: median latency, which is a measure of how long any given transaction might have to wait to be processed and the amount of elapsed time, which is a measure of how long it will take to complete either the read or insert session. Latencies at the

95th and 99.9th percentile were also reported to provide an idea of how the system might function under duress, akin to the "millions-of-hits" scenario mentioned earlier. Also, the session time (time it took to run a specific trial) was reported as well. 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 one million records to a maximum of 15 million records. A single client against a Cassandra database containing eight nodes generated the initial tests detailed in the section below.

## RESULTS

The results are reported below in Table 1. These results indicate that the eight-node Cassandra database did a pretty good job of scaling from one million reads/inserts to ten million reads/inserts. While data was obtained for inserts at the fifteen million-level, the same experiment conducted at that level for reads generated so many errors that the data was not available. In terms of session time (reported as m:ss) as the workload increased, as one would expect, so did the session time. In fact, the five million times are about five times greater than the one million times results, and that linear pattern generally holds true at the ten and fifteen million levels.

Interestingly, the median latency remained in the 1.4 to 1.8 second range across all intensity levels. Further, the results obtained up to the 95th percentile were encouraging which ranged from 2.6 to 3.2 seconds. For example, if one's target latency was three seconds or less (as mentioned earlier as a threshold of end user expectations) then the current configuration would be able to handle that up to ten million records with a 95% confidence factor. However, there are numerous outliers at the 99.9th percentile at all intensity levels. All values exceeded 20 seconds and the decay was really noticeable at the fifteen million level where a value greater than 800 seconds was observed. For many applications (such as e-commerce) having even one outlier of the magnitudes observed at the 99.9th percentile would be unacceptable. In e-commerce customers expect a maximum wait time of three to five seconds, if that threshold is exceeded they may simply migrate to a competitor never to return.

In evaluating Table 1, it is important to understand that there are multiple processors available to handle the read/insert requests. Therefore, some fairly substantial workloads can be accommodated by the system provided that the queues do not get overloaded. In the case of the four workload density trial, only the last test (fifteen million reads) experienced queue overloading issues. The first three workload levels all exhibited similar values in the median latency and 95th percentile latencies. However, the 99.9th percentile latencies are lower for the one million, and roughly the same for the five and ten million record tests. The real difference is in the session time which increases significantly, but not quite linearly as the workload increases. So in other words, the service rate remains relatively constant but how long it takes to service the request increases with the load.

**TABLE 1: MEDIAN LATENCIES/SESSION TIMES FOR A CASSANDRA DATABASE AT VARIOUS RECORD INTENSITIES**

| | one million | | five million | | ten million | | fifteen million | |
|---|---|---|---|---|---|---|---|---|
| | insert | read | insert | read | insert | read | insert | read |
| median latency | 1.4 | 1.7 | 1.6 | 1.7 | 1.6 | 1.8 | 1.7 | NA |
| latency 95 percentile | 2.6 | 2.9 | 2.8 | 3 | 2.8 | 3 | 3.2 | NA |
| latency 99.9 percentile | 38 | 22.8 | 81 | 91.7 | 73.8 | 91.1 | 824.4 | NA |
| session time | 0:58 | 0:54 | 5:01 | 4:57 | 9:53 | 9:16 | 17:00 | NA |

While generally speaking, the latencies do not differ greatly for reads versus inserts in a given category, inserting was typically quicker. In part, the slight difference between reads and inserts, as can be seen in Figure 1, may be related to the buffering process. For reads, each time a new and different record was read in a random fashion, just that record could be cached. However, with inserts the whole new record could be placed in a buffer without waiting and then actually inserted at the appropriate disk location. Also, the overhead of transferring data in a distributed node environment with a one-gigabit per second network connection may have exacerbated the latencies of the read process, which could not be treated as a "block" of data.

There was no inherent automated tuning process configured within the Cassandra implementation, so even though millions of records were processed, the system did not learn how to improve upon the basic read process. In other words, some databases allow for dynamic growth while others do not. For example, with Microsoft SQL Server this is handled most of the time as a percentage of the current size. So as the size of the data block increases there is an expected delay that occurs. The bigger the data block gets, the longer the gap and more infrequent the delay occurs. So therefore, database learning can also be a factor in how efficiently it will operate. Initial queries can thus take longer as the query optimizer creates more efficient plans to effectively fulfill them. As the plans developed by the optimizer improve, the performance can and should also improve. This is a concept that merits future investigation into how overall database efficiency may be affected.

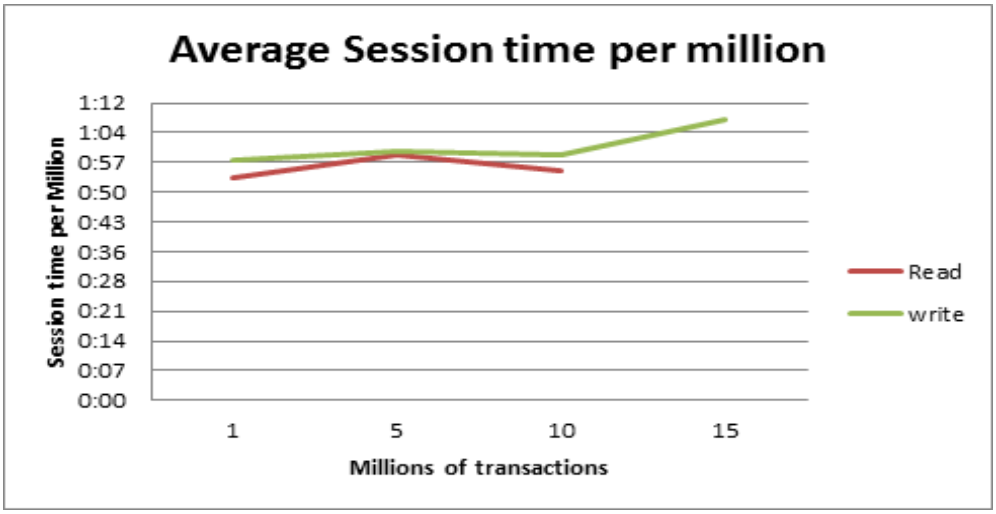


**FIGURE 1: GRAPHICAL REPRESENTATION OF TABLE 1**

## DISCUSSION

From the data contained in Table 1 it is clear that generally speaking the Cassandra distributed database scales fairly well up to the ten million-intensity levels. At the fifteen million level the read experiment generated so many errors that it was unusable and a 99.9$^{th}$ percentile latency on the insert level of 800+ seconds indicated that the configuration was not adequate for inserts as well. The observed latencies at the one through ten million levels were encouraging, but need to be analyzed in the scope of a target response time. These targets may vary depending on the application supported. For example, if the application being considered is e-commerce then response time is critical. In this case, a total end-to-end response time of three seconds or less is critical to meet a customer's expectations, though the quicker the better. If this target metric is not met then customers may get impatient and migrate to another site, perhaps never to return. Therefore, a profit making enterprise would be well advised to carefully evaluate the performance gains Cassandra could provide. Based on the data reported herein, the metrics need to be evaluated in regard to the number of transactions that will exceed the desired threshold.

Given the data observed above one million records and a maximum database latency target of three seconds or less the metrics for both reads and inserts meet the target on the median and 95$^{th}$ percentile metrics. However, at this level the target is greatly exceeded at the 99.9$^{th}$ percentile. Because computer systems are dynamic in nature and workload for the most part does not follow a normal distribution it is very difficult to completely eliminate the outliers that are measured by the 99.9$^{th}$ percentile in this paper (Guster, 2002; Guster, Robinson, & Safonov 2005; Guster, Robinson, & Sundheim 2008; Rabl et al., 2012). Therefore, tuning a database can be tricky. Is it acceptable to have less than 40 out of one million transactions exceed the target by a factor of 10? Certainly a company needs to weigh the added cost. Perhaps, it might spend $50,000 to tune it and reduce the number of outliers from less than 40 to less than 20, would this be an acceptable return on investment? That would be for the company to decide. Further, as previously stated,

computer systems are dynamic and need to be monitored. It might not be unusual to observe an increase in outliers at a future date, which can often happen when software elements in the system have been upgraded or patched. While the magnitude of the median latencies observed at the one million level seem in line with a three second target, with an application such as an e-commerce website the more important metric is actually end-to-end delay.

In other words, one needs to take into account all of the components required to complete a transaction between a client and a server. In other words, one needs to take into account all of the components required to complete a transaction between a client and a server. A good representative example of this model is offered by Fleming (2004). This model can be summarized as follows: User Application Delay + CPU Delay on the Local Computer + Net work interface card (Ethernet) Delay on the Local Computer + Delay passing through the Network Switching devices + Delay passing across the network itself + Delay passing through the Network Switching devices on the receiving side + Net work interface card (Ethernet) Delay on the receiving Computer + Application service Delay + Disk controller Delay + Delay in the Disk Read Process, (…then the delay in traversing the path in reverse for the reply). Because there are many components which interact with one another even a small delay value like ~.0025 seconds can be significant and therefore, one needs to look at the total response time model which is quite complex and involves a number of components.

In another experimental study that dealt with reducing latency through adding nodes addressed the concept of end-to-end delay (Guster, O'Brien, & Lebentritt, 2013). While the Internet provides pretty good performance, considering its size and scope, one can expect delays of various magnitudes based on its workload at the time the transaction is consummated. For example, given that the network delay on the Internet in the U.S. could be described by the median value .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 play. 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 one second (.8192) (Guster et al., 2013).

Besides the potential to scale effectively while maintaining reasonable performance the capabilities of Cassandra to support fault tolerance are quite useful. Furthermore, Cassandra is able to accomplish this feat by performing on inexpensive commodity hardware. Its design also allows for easy tuning of the original configuration. This situation is ideal for small/medium size companies that want to adopt Cassandra for its cost effectiveness and ease of use. Because of the ease that replication can be invoked it would be effective to use Cassandra as a central core 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 (Guster, et al. 2013; Cockcraft, 2011).

## CONCLUSIONS AND FUTURE STUDY

The data contained herein, was designed to ascertain how an eight-node Cassandra database would scale at high intensity workloads. Although the results were interesting and generally indicated respectable scaling there are still numerous complexities that need to be explored when implementing Cassandra. First, for the sake of simplicity in this study a single client generated the workload, which is not realistic. Using multiple clients would certainly be more realistic and certainly would alter the intensity and distribution of the incoming workload. Second, this data was collected using a configuration that featured only one replica. Given the distributed nature of most companies' data profile research is needed that addresses multiple nodes distributed world-wide. Third, while the workload was fairly intense, research is needed that will look at a particular workload and then try to tune the existing test bed to better accommodate that intensity. Fourth, the number of nodes was limited to eight. Although based on the latencies observed this appeared to be a logical starting point, in a production world one would expect the number of nodes to far exceed this value. While the results obtained herein answer some scaling related questions, there are still numerous unanswered questions. Therefore, future research will need to address configuration, workload and tuning variables to ascertain the effectiveness of Cassandra in various applications.

## REFERENCES


Abadi, D. J., (2012). Consistency tradeoffs in modern distributed database system design: cap is only part of the story. *Computer*, 45 (2), 37-42.

Afyouni, H. A. (2006). *Database security and auditing: Protecting data integrity and accessibility*. Boston, MA: Course Technology, Cengage Learning.

Apache Software Foundation (2009). *Cassandra*. Retrieved from http://cassandra.apache.org

Bernstein, P., Rothnie, J. B., Goodman, N., & Papadimitriou, C. A. (1978). The concurrency control mechanism of sdd-1: a system for distributed databases (the fully redundant case). *IEEE Transactions on Software Engineering*, 4 (3), 154-168.

Bonifati, A., Panos K. C., Aris M. O., & Kai-Uwe S., (2008). Distributed databases and peer-to-peer databases: past and present. *ACM SIGMOD Record Archive*, 37 (1), 5-11. ACM New York, NY, USA doi>10.1145/1374780.1374781.

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.

Cockcraft, A. (2011). Migrating netflix from a datacenter oracle to global cassandra. *Cassandra Summit Talk 2011*. Retrieved http://www.slideshare.net/adrianco/migrating-netflix-from-oracle-to-global-cassandra.

DataStax Corporation. (2013). Benchmarking top nosql databases. Retrieved from http://datastax.com/wp-content/uploads/2013/02/WP-Benchmarking-Top-NoSql-Databases.pdf.

Elnikety, S., Tracey, J., Nahum, E., and Zwaenepoel, W. (2004). A method for transparent admission control and request scheduling in e-commerce web sites. *Proceedings of the 13th International Conference on World Wide Web*, 276-286.

Featherston, D. (2010). Cassandra: Principles and applications. Retrieved from http://disi.unitn.it/~montreso/ds/papers/Cassandra.pdf.

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.

Guster, D. C. (2002). A comparison of stochastic models for interarrival times of packets in a computer network. *Optimal Information Modeling Techniques*: IRM Press.

Guster, D. C., Robinson, D. H., & Safonov, P. I. (2005). Packet inter-arrival distributions in computer network workloads. *Encyclopedia of Information Science and Technology*. Hersey, PA: Idea Group.

Guster, D. C., Robinson, D., & Sundheim, R. A. (2008). Evaluating computer network packet distributions. *Encyclopedia of Information Science and Technology*. Hersey, PA: Idea Group.

Guster, D. C., O'Brien, A., & Lebentritt, L. (2013). Can a decentralized structured storage system such as cassandra provide an effective means of speeding up web access times. *Proceedings of Midwest Instructional Computing Symposium*.

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.

Kanitkar, V. (2000). Collaborative and real-time transaction processing techniques inclient-server database architectures. *Polytechnic University*, 61 (04B), 2036.

Kanitkar, V. and Delis, A. (2002). Distributed query processing on the grid. *IEEE Transactions on Computers*, 51 (3), 269-278.

Lakshman, A. and Malik, P. (2010). Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44 (2), 35-40.

Rabl, T., Sadoghi, M., Jacobsen, H-A., Gomez-Villamor, S., Muntes-Mulero, V., & Mankovskii, S. (2012). Solving big data challenges for enterprise application performance management. Retrieved http://vldb.org/pvldb/vol5/p1724_tilmannrabl_vldb2012.pdf.

Ozsu, M. T., & Valduriez, P. (1991). Distributed database systems: where are we now? *Computer*, 24 (8), 68-78.

Otey, M. E., Parthasarathy, S., Wang, C., Veloso, A., & Meira, W. (2004). Parallel and distributed methods for incremental frequent itemset mining. *IEEE Transactions on Systems, Man & Cybernetics: Part B*. 34 (6), 2439-2450.

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

Xu, W., Jing L., Yongwei W., Xiaomeng, H. & Guangwen, Y. (2008). VDM: virtual database management for distributed databases and file systems. *Seventh International Conference on Grid and Cooperative Computing*, 309- 315.

Zimmermann, R., Wei-Shinn, K., & Wei-Cheng, C., (2004). Efficient query routing in distributed spatial databases. GIS '04 Proceedings of the 12th annual ACM: *International Workshop on Geographic Information Systems*, 176 - 183 ACM New York, NY, USA ©2004 table of contents ISBN:1-58113-979-9 doi>10.1145/1032222.1032249.

JOURNAL OF INTERNATIONAL BUSINESS DISCIPLINES

Volume 9, Number 1 May 2014

**Published By:**
International Academy of Business Disciplines and Frostburg State University


ISSN 1934-1822 WWW.JIBD.ORG