JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

Benchmarking Correctness of Operations in Big Data Applications

Gannoju Vinod Kumar
Department of Computer Science and Engineering,
Sphoorthy Engineering College,
Jawaharlal Nehru Technical University, Hyderabad

ABSTRACT

The past several years have seen an increase in data stores with innovative design choices that may compromise an application's operational accuracy in order to improve performance, due to the large range of big data applications. This paper outlines our ongoing efforts to develop a framework that produces a part of validation. An application's characteristics are fed into the framework. Its output is a validation module that may be used to gauge how much unpredictable data a data store produces by plugging it into an application or a benchmark. This paper provides a synopsis of the BG benchmark by identifying its strengths and limitations in our daily use cases. The identified limitations shape our research activities and the obtained solutions shall be incorporated into future BG releases.

A INTRODUCTION

In order to manage the ever-increasing volume and variety of data created by big data applications, there has been an explosion of new data stores with diverse architectures and design choices. Systems and services with innovative assumptions are still being contributed by academia, cloud service providers like Google and Amazon, social networking sites like Facebook and LinkedIn, and the computer industry. In the year 2010 Rick Cattell assessed 23 systems [1], and 10 more1 have come to our attention since then. "NoSQL" data storage are a common term for several of these emerging systems. Although there is no universally accepted definition of NoSQL, the systems covered in [1] typically lack strong consistency guarantees, or ACID transactional features. They can choose the Basically Available, Soft state instead.Novel data stores with a wide range of architectures and design choices have proliferated to handle t he ever-increasing volume and diversity of data generation.

Novel data stores with a wide range of architectures and design choices have proliferated to handle the ever-increasing volume and diversity of data generBenchmarking frameworks have been used to quantify a variant of the first two measures [9, 10, 6, 12].

The sole benchmark available for assessing proper operation execution in support of interactive social networking operations is BG [6, 12]. In order to achieve a work-in-progress, this document abstracts BG's validation process.

Framework that is modular and customizable and assesses the accuracy of various applications. The features of an application or a benchmark tailored to a particular application serve as the framework's input. The framework produces a module that can be integrated into the benchmark or the application to measure the quantity of uncertain data resulting from improper operation execution. One possible reason for an inaccurate execution could be inconsistent replicas of a data item.

Novel data stores with a wide range of architectures and design choices have proliferated to handle the ever-increasing volume and diversity of data generOne or more data items may have their attribute values updated by an application's operations. A database error could lead to surprising results for all ensuing reads if the values generated by these operations are not appropriately saved. A relational database management system (RDBMS) enhanced with memcached is used to observe the amount of unpredictable data, which is plotted against the system load simulated by a requests are issued by a certain number of threads, T. There is a set time to live (TTL) for the cached entries, which can range from 30 to 120 seconds. The proportion of requests with response times less than 100 milliseconds is indicated by the numbers in red. As a result of answering queries with larger TTL values, this proportion rises for various system loads.

Novel data stores with a wide range of architectures and design choices have proliferated to handle the ever-increasing volume and diversity of data generOne or more data items may have their attribute values updated by an application's operations. A database error could lead to surprising results for all ensuing reads if the values generated by these operations are not appropriately saved. A relational database management system (RDBMS) enhanced with memcached is used to observe the amount of unpredictable data, which is plotted against the system load simulated by a requests are issued by a certain number of threads, T. There is a set time to live (TTL) for the cached entries, which can range from 30 to 120 seconds. The proportion of requests with response times less than 100 milliseconds is indicated by the numbers in red. As a result of answering queries with larger TTL values, this proportion rises for various system loads.

Novel data stores with a wide range of architectures and design choices have proliferated to handle the ever-increasing volume and diversity of data generOne or more data items may have their attribute values updated by an application's operations. A database error could lead to surprising results for all ensuing reads if the values generated by these operations are not appropriately saved. A relational database management system (RDBMS) enhanced with memcached is used to observe the amount of unpredictable data, which is plotted against the system load simulated

Novel data stores with a wide range of architectures and design choices have proliferated to handle the ever-increasing volume and diversity of data generOne or more data items may have their attribute values updated by an application's operations. A database error could lead to surprising results for all ensuing reads if the values generated by these operations are not appropriately saved. A relational database management system (RDBMS) enhanced with memcached is used to observe the amount of unpredictable data, which is plotted against the system load simulated by a requests are issued by a certain number of threads, T. There is a set time to live (TTL) for the cached entries, which can range from 30 to 120 seconds. The proportion of requests with response times less than 100 milliseconds is indicated by the numbers in red. As a result of answering queries with larger TTL values, this proportion rises for various system loads.

Novel data stores with a wide range of architectures and design choices have proliferated to handle the ever-increasing volume and diversity of data generOne or more data items may have their attribute values updated by an application's operations. A database error could lead to surprising results for all ensuing reads if the values generated by these operations are not appropriately saved. A relational database management system (RDBMS) enhanced with memcached is used to observe the amount of unpredictable data, which is

plotted against the system load simulated by a requests are issued by a certain number of threads, T. There is a set time to live (TTL) for the cached entries, which can range from 30 to 120 seconds. The proportion of requests with response times less than 100 milliseconds is indicated by the numbers in red. As a result of answering queries with larger TTL values, this proportion rises for various system loads.

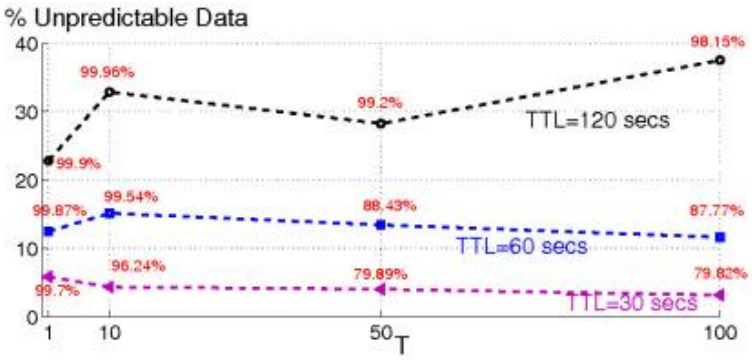

Figure 1: Amount of unpredictable data observed with a relational database management system (RDBMS) augmented with memcached.

Novel data stores with a wide range of architectures and design choices have proliferated to handle the ever-increasing volume and diversity of data generOne or more data items may have their attribute values updated by an application's operations. A database error could lead to surprising results for all ensuing reads if the values generated by these operations are not appropriately saved. A relational database management system (RDBMS) enhanced with memcached is used to observe the amount of unpredictable data, which is plotted against the system load simulated by a requests are issued by a certain number of threads, T. There is a set time to live (TTL) for the cached entries, which can range from 30 to 120 seconds. The proportion of requests with response times less than 100 milliseconds is indicated by the numbers in red. As a result of answering queries with larger TTL values, this proportion rises for various system loads.

B UNPREDICTABLE READS

From the standpoint of a client issuing them, consider the simultaneous execution of many concurrent write operations with a read operation (see Figure 2). The result obtained by the read operation R1 relies on how a data store is configured, assuming that all of these actions pertain to the same data item Di and are complete.

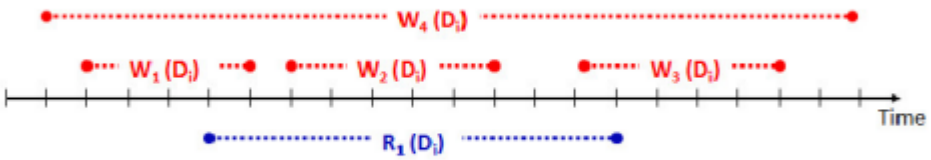

Figure 2: There are four ways for a write of D_i to overlap a read of D_i.

executes them in a serialized manner. A serial execution of the concurrent operations in isolation is one possibility [18].

Consequently, the value that R1 retrieves is a range of values that can be obtained based on how the data store serializes the read operation in relation to the write operations. It is possible that the read operation in Figure 2 was serialized.

Either before all write operations, after all write operations, or gradually after each write operation (taking into account all the possible combinations of W4 with other write operations). The data store generated an accurate value if R1 returns a value that is present in the set. If not, R1 has seen data that is unpredictable.
There are two possible methods for validation: online and offline. An offline technique would measure the amount of unpredictable data after a benchmark finishes generating its stated workload, whereas an online strategy would indicate the accuracy of a value created for a read operation immediately after it completes does not measure the quantity of erratic data that is collected during the trial. A method can apply one of the two extremes or a hybrid approach, as described in Section C, which provides the quantity of uncertain data after a certain period of time depending on the resources allotted.
Validation needs to support both custom and primitive data types in order to plug-in to a variety of application-specific benchmarks. Among the primitive data types are Char, String, Boolean, int, and so forth. Calendar events and other user-defined objects are examples of custom data types.

Two main inputs must be customizable for the validation approach in order to calculate the amount of unexpected data. First, a method for distinguishing a data item from others, read and write functions that modify 4 operations.
Data items, and how write operations alter a data item's status from one that is valid to another. One example of a data item in a social networking benchmark would be a member profile located via the member's unique identifier. View Profile (VP) and Accept Friend Request (AFR) are two examples of read and write operations, respectively. The write operation increases each member's friend count by one when Member A issues the AFR operation to verify friendship with Member B.

Custom data types are subject to operation dependence and data item specifications. For instance, if member A is displaying an array-type feed, then when member A writes to unfriend B, B's news shouldn't show up in member A's feed [15]. How to specify things in an intuitive way is a challenge.

A crucial component of our project is creating user interfaces that are effective. Second, there needs to be enough input data such that the validation procedure can calculate the value that a read operation ought to have seen. The starting value of a data item is a crucial input. Analytical models could be one form of this input.

C An Implementation

Our development and use of BG's validation methodology served as the foundation for the abstraction in Section
B.
BG is a scalable benchmarking system that measures the quantity of uncertain data following the completion of the benchmark by implementing validation as an offline technique. This choice reduces the quantity of resources needed by BG to provide a load that makes the most of a data store's resources. An online version would allow an experimenter to see the volume of erratic data together with other online metrics that have been reported (such response time and throughput). Nevertheless, more CPU and memory may be needed.

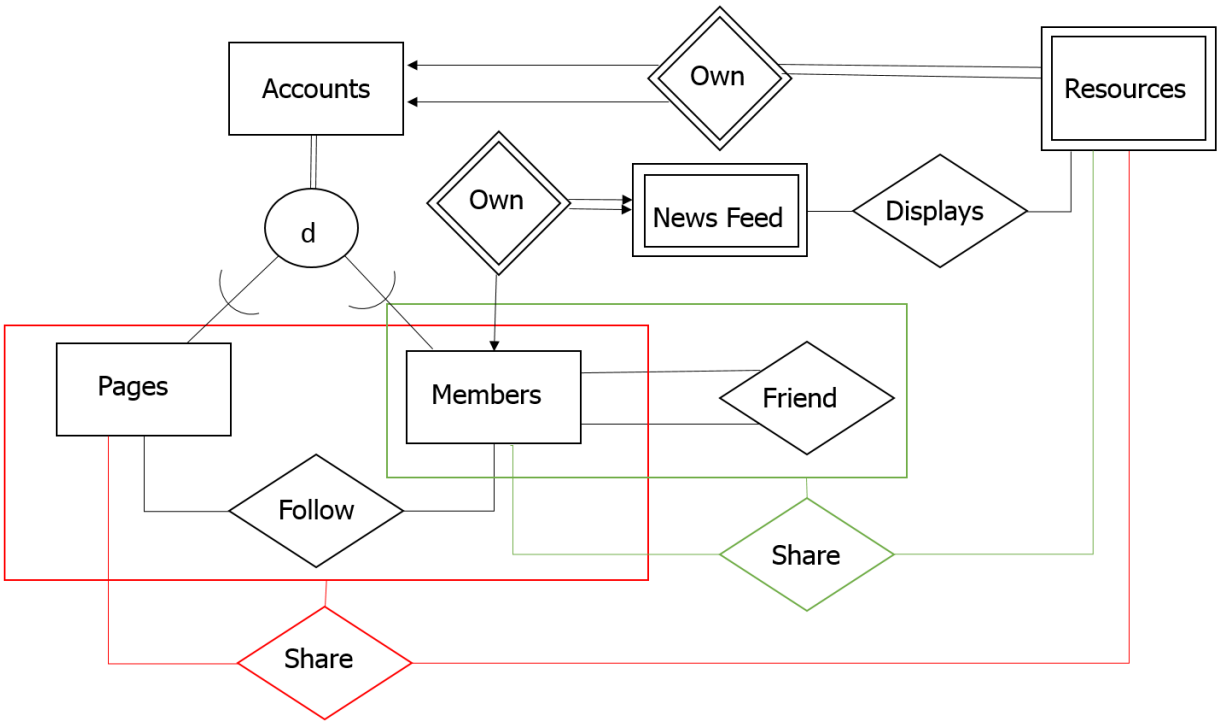

Figure 4: Conceptual design of BG's feed following.

In order to give the validation module with the specifics of the actions carried out during the benchmarking phase, today's BG generates a stream of log records [6, 12]. These contain the start and finish timestamps of the operations issued by the BG benchmark and are divided into read and write log records.

The Records from read logs preserve the value that was observed from a data store. The write log entries show whether an attribute of a manipulated data item has changed or has a new absolute value. These log records are used in conjunction with the database's initial state during the offline validation procedure to determine how many unpredictable reads there are. The social graph's starting state is determined by a set number of members, friendships for each member,

One way to inform the validation process about the actions carried out by a benchmark is to generate these log records. It is easy to route BG's stream of log records to a procedure that would carry out the semi-online2 validation. One could picture a dataflow. MapReduce3 infrastructure that gradually improves the amount of uncertain data computing.

A subset of the design options are implemented in the validation technique as it is currently used. This implementation is meant for a typical use case scenario; it is not meant to be used in all situations. For example, the current approach makes the assumption that the PC carrying out validation4 has enough memory to hold all of the write log records. High capacity data stores that process millions of operations per second are incompatible with this.

The validation causes the PC's memory to run out, which results in a thrashing operating system. Our current studies with middlewares like KOSAR [16, 17] demonstrate this. An experimentalist might not be willing to wait hours to get an exact measurement of the amount of uncertain data, even with enough memory.

The Runtime Configuration component, as its name implies, sets up the modules that are used during the validation stage. This part is utilized just once. The components used by the validator during runtime are specified in its output. Based on the following input parameters, its result is produced:

• How much RAM ought to be set aside for the validation stage? In order to avoid the validation phase using up all of the memory during the processing of read log records, a certain amount of write log records might be staged in memory before being processed.
2.Not entirely online since there could be a lag between the time the log records are generated and the processing time.

Since the records can be divided according to the identity of the data objects that a log references, 3MapReduce makes sense.
In the validation process, the accuracy of the value generated by a data storage is assessed. It can use a probabilistic model with a predetermined confidence level or a deterministic model to ascertain 7 accuracy. These models ought to be plug-and-play software modules in theory. To calculate correctness, they might need to know the starting state of the data items and a set of precise write operations. As a result, there's a chance that this stage depends on the Runtime configuration and Log processing stages.

D Summary

Our ongoing efforts to develop a general-purpose framework for assessing the accuracy of operations in big data applications are presented in this study. It measures the quantity of random data generated by innovative data storage architectures that could jeopardize operation correctness because of unfavorable race situations and delayed update propagation. An experimentalist could create application-specific benchmarks and gauge the volume of unpredictable data generated by a data source using such a transformational framework. Applications in data science [11] with a variety of data sets and use case scenarios benefit most from it.

References

[1] R. Cattell. Scalable SQL and NoSQL Data Stores. SIGMOD Rec., 39:12–27, May 2011.
[2] S. Ghandeharizadeh and J. Yap. Gumball: A Race Condition Prevention Technique for Cache Augmented SQL Database Management Systems. In Second ACM SIGMOD Workshop on Databases and Social Networks, 2012.
[3] P. Gupta, N. Zeldovich, and S. Madden. A Trigger-Based Middleware Cache for ORMs. In Middleware, 2011.
[4] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, Harry C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung, and V. Venkataramani. Scaling memcache at facebook. In Presented as

part of the 10th USENIX Symposium on Networked Systems Design and Implementation, pages 385–398, Berkeley, CA, 2013. USENIX.

[5] H. Wada, A. Fekete, L. Zhao, K. Lee, and A. Liu. Data Consistency Properties and the Trade-offs in Commercial Cloud Storages: The Consumers' Perspective. In CIDR, 2011.

[6] S. Barahmand and S. Ghandeharizadeh. BG: A Benchmark to Evaluate Interactive Social Networking Actions. Proceedings of 2013 CIDR, January 2013.

[7] P. Bailis and A. Ghodsi. Eventual Consistency Today: Limitations, Extensions, and Beyond. Communications of the ACM, May 2013.

[8] P. Bailis, S. Venkataraman, M.J. Franklin, J.M. Hellerstein, and I. Stoica. Quantifying Eventual Consistency with PBS. The VLDB Journal, pages 1–24.

[9] M.R. Rahman, W. Golab, A. AuYoung, K. Keeton, and J.J. Wylie. Toward a Principled Framework for Benchmarking Consistency. In Proceedings of the Eighth USENIX Conference on Hot Topics in System Dependability, HotDep'12, pages 8–8, Berkeley, CA, USA, 2012. USENIX Association.

[10] S. Patil, M. Polte, K. Ren, W. Tantisiriroj, L. Xiao, J. Lopez, G. Gibson, A. Fuchs, and B. Rinaldi. YCSB++: ´ Benchmarking and Performance Debugging Advanced Features in Scalable Table Stores. In Cloud Computing, New York, NY, USA, 2011. ACM.

[11] A. Talukder, C. Greenberg. Overview of the NIST Data Science Evaluation and Metrology Plans. In Data Science Symposium, NIST, March 4-5, 2014.

[12] S. Barahmand. Benchmarking Interactive Social Networking Actions. Ph.D. thesis, Computer Science Department, USC, 2014.

[13] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking Cloud Serving Systems with YCSB. In Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC '10) , 2010.

[14] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, W. Vogels. Dynamo: Amazon's Highly Available Key-value Store. In Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles (SOSP '07), 2007.

[15] S. Barahmand, S. Ghandeharizadeh and D. Montauk. Extensions of BG for Testing and Benchmarking Alternative Implementations of Feed Following. In Proceedings of the SIGMOD Workshop on Reliable Data Services and Systems (RDSS), 2014.

[16] S. Ghandeharizadeh. KOSAR: A Game Changer for SQL Solutions. Mitra LLC, 2014.

[17] S. Ghandeharizadeh. KOSAR: An Elastic, Scalable, Highly Available SQL Middleware. USC DBLAB Technical Report 2014-09.

[18] J. Gray and A. Reuter. Transaction Processing: Concepts and Techniques. Morgan Kaufmann Publishers Inc., 1992