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

An Overview of the BG Social Networking Benchmark in Mid-Flight

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

ABSTRACT

A data store's ability to handle interactive social networking actions, such viewing a member's profile, accepting friend requests, and so on, is measured by a benchmark called BG. The expansion of data stores from many academic and industrial sources, such as social networking firms as Voldemort by LinkedIn, is the driving force behind it. BG is intended to give a system architect insights into alternative design principles, such as using a weak consistency technique rather than a strong one, varying physical data models, such as JSON and relational, factors influencing a data store's ability to scale vertically and horizontally, and the CAP theorem's consistency versus availability tradeoff, among other things.

A INTRODUCTION

David Patterson notes in a piece that was published in the Communications of the ACM in July 2012 that when a discipline has strong standards, disagreements are resolved and the discipline advances quickly [16]. Now, we own a multitude of data storage and service architectures, but only a small number of benchmarks to support their numerous assertions. The computer industry, social networking sites like Facebook and LinkedIn, cloud service providers like Google, and academia all keep bringing new systems and services with innovative architectures to the table.

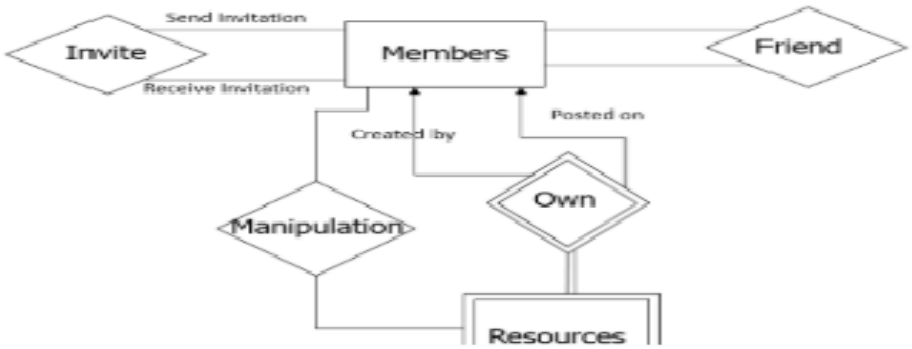

Figure 1: Conceptual design of BG's database.

Rick Cattell conducted a review of 23 systems in 2010 [7], and since then, 10 additional ones have come to our attention. A "gaping hole" with few benchmarks to support the promises made by the various systems was noted by Cattell in his survey.

A social networking benchmark called BG [2] (see http://bgbenchmark.org) has been created and put into use to solve some of the holes that are too big to be filled by a single benchmark. BG's workload consists of actions that either read or write a small amount of data from big data, typically termed simple operations [10].

The BG of today is made for interactive reaction speeds and large throughput data repositories. Long-term goals include expanding BG using sophisticated analytics that and presumptions. Rick Cattell conducted a review of 23 systems in 2010 [7], and since then, 10 additional ones have come to our attention. Cattell's survey revealed a "gaping hole" where there are few benchmarks to support the promises made by the various systems. To address some of the holes that are too big to be filled by a single benchmark, we have created and implemented a social networking benchmark called BG [2] (see http://bgbenchmark.org).

Simple operations, as they are commonly called, are actions that either read or write a little quantity of data from huge data [21, 10]. The BG of today is made for interactive reaction speeds and large throughput data repositories. Long-term goals include expanding BG using sophisticated analytics.

BG Social Actions	Type	Very Low (0.1%) Write	Low (1%) Write	High (10%) Write
View Profile (VP)	Read	40%	40%	35%
List Friends (LF)	Read	5%	5%	5%
View Friends Requests (VFR)	Read	5%	5%	5%
Invite Friend (IF)	Write	0.04%	0.4%	4%
Accept Friend Request (AFR)	Write	0.02%	0.2%	2%
Reject Friend Request (RFR)	Write	0.02%	0.2%	2%
Thaw Friendship (TF)	Write	0.02%	0.2%	2%
View Top-K Resources (VTR)	Read	40%	40%	35%
View Comments on Resource (VCR)	Read	9.9%	9%	10%
Post Comment on a Resource (PCR)	Write	0%	0%	0%
Delete Comment from a Resource (DCR)	Write	0%	0%	0%

Table 1: Three mixes of social networking actions.

Microsoft Azure is a provider of cloud services. Use Hibernate as a framework for object relational mapping (ORM). ~ Memcached, EhCache, Twemcache, and KOSAR are examples of cache-augmented data storage.

Table 1's first column lists the eleven acts that make up BG. The actions' names are self-explanatory. Every binary action that refers to a member requires two member ids as input. For instance, the member who is seeing a profile and the member whose profile is being viewed are identified by the two member ids provided with the View Profile action. When a resource is consumed, one of two things happens: either the resource and its comments are read, or a comment on that special resource is modified. A data store may be read or written by the various actions, as indicated by Table 1's second column. For the curious reader, a comprehensive explanation of each step may be found in [2].

Table 1's final three columns display three distinct workloads that represent extremely low, low, and high percentages of write actions. (Facebook claims that read activities make up over 99% of their workload [9].)

For every action in a workload, a predetermined proportion of occurrence is specified in each column. We often use the three workloads that have been demonstrated in our experiments. Each of the three symmetric workloads results in roughly the same amount of verified and pending friendships at the end of the trial as there were at the start. The frequency of the following interactions—Invite Friend, Accept Friend Request, Reject Friend Request, and Thaw Friendship actions—affects the quantity of these relationships.

A stateful benchmark that produces legitimate actions is called BG. For instance, it only allows Member A to befriend Member B when they are not already friends. It does this by keeping an image of the social graph stored in its memory. BG makes use of this representation to guarantee that the resources and members it emulates simultaneously are distinct at any given moment.
The capacity of BG to measure the quantity of unpredictable (stale, inconsistent, and erroneous) data generated by a data store is a novel characteristic. A data store is assessed by BG for a workload with a specified SLA. During instance, a SLA can stipulate that during 10 minutes, 95% of activities must be completed in less than 100 milliseconds and with no more than 0.01% of the data being unpredictable. BG has a search heuristic.

In order to compare a relational representation of a social graph with its JSON representation [5], quantify the tradeoffs associated with alternative consistency techniques for a cache augmented relational data store [12], and other tasks, we have used BG to investigate the design and implementation of novel architectures for data intensive applications. We have found many BG design restrictions in these application cases. These inform the way we do research to keep BG as a cutting-edge standard. The majority relate to the new aspects of BG that set it apart. We go over each of these in turn below, describing in Section B how BG generates scalable requests, Section C how it emulates socialites both closed and open, and Section D how it rates socialites.

Part D, its verification stage in Section E, and further measures in Section F. We present our long-term future research in Section G.

B Scalability

Because BG uses a shared-nothing design and can expand to a high number of nodes, its request creation rate is not limited by a single node's CPU, memory, or network capabilities. One coordinator and one or more clients—referred to as BGCoord and BGClient, respectively—make up its software architecture. A multi-threaded BGClient may completely utilize all cores in our 8 core CPU trials, provided that the data store can process requests at the rate that BG generates and the client part of the data store is not affected by the convoy phenomenon
[6].
If a data store's client component restricts vertical scaling and there is enough memory available,

To make use of every core, several instances of BGClients can run on a single node. In order to grow horizontally, BG runs several BGClients on various nodes. In order to calculate the SoAR of a data store, BGCoord is in charge of starting the BGClients, keeping track of their progress, compiling their findings at the conclusion of an experiment, and combining the results.
The BGClient instances do not synchronize after they are launched; instead, they generate requests individually. The two ideas that follow enable this to happen. A benchmark social graph is first partitioned into disjoint sub-graphs by a BGClient using a decentralized partitioning approach, where { is the number of BGClients. A sub-graph is assigned to a BGClient, allowing it to create requests that only reference members of that sub-graph.

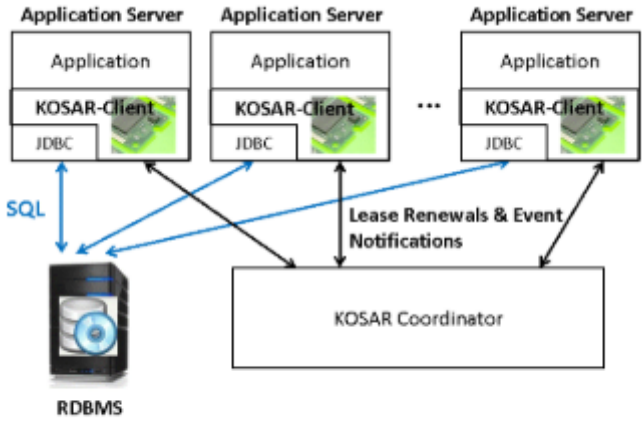

Figure 2: A data intensive architecture using KOSAR.

As the data repository does not recognize this division, even though the data created and kept in the data storage does match the disjoint graphs. Each sub-graph can be thought of as a province whose residents are limited to performing BG's behaviors with one another. As a result, residents of various provinces are unable to see each other's profiles or add friends.

BG utilises a unique decentralised Zipfian distribution implementation, called D-Zipfian [3, 24], to guarantee that the allocation of requests among the various members is not influenced by {. As a result, the distribution of access between one and multiple nodes is the same. By using D-Zipfian and social graph partitioning, BG can use nodes to produce requests without waiting for coordination till the end of experiment, see [2, 4, 3] for details Even if BG scales to a large number of nodes, its two concepts might not be able to fairly assess all data storage.

Take into consideration, for illustration, the architecture shown in Figure 2, which extends an application with a cache like KOSAR or EhCache [12]. The KOSAR coordinator, or KOSAR-Client for short, is the component of this caching technology. It keeps track of which application server has cached a copy of a data item in its KOSAR JDBC wrapper. The KOSAR coordinator is notified of the impacted data item when one application server modifies a copy of the data item by its KOSAR-Client. A copy of this data item that is

stored in the KOSAR-Client of other application servers is then invalidated by the KOSAR coordinator. With a distorted member access pattern and a A centralized KOSAR coordinator could become the bottleneck and control system performance for workloads with low read to write ratios. The two previously mentioned ideas that BG uses don't result in the development of such a bottleneck. To clarify, since each application server is assigned a unique sub-graph that is separate from all other sub-graphs, it refers data objects that are unique to itself. Therefore, BG does not use the KOSAR coordinator's ability to notify KOSAR-Client of another application server if an application server updates a cached data item.

The aforementioned restriction is being addressed by expanding BG to include BGClients with a single social graph. Hashing partition members and resources across BGClients is the fundamental idea. Every BGClient uses the original Zipfian distribution (rather than the D-Zipfian) to create member ids and is aware of the hash function. When a BGClient references a data item that is not part of its allocated partition, it gets in touch with the BGClient that is the owner of the referred data to lock that data item for usage only by that BGClient. and to ascertain whether the planned activity is feasible. If the referred data item does not already have a lock and the action is feasible, grants the lock request, allowing to move forward with creating a request with the identified data item to the data store.

How should the framework resolve a conflict when fails to issue an exclusive lock2 to the referred data item because there is already a lock in place? Here are three possible outcomes. Initially, it might obstruct until the data item mentioned above is made available. Secondly, it might give an error message to \ , telling them to try again with an alternative member or resource ID. Thirdly, it might just stop this action and start a whole new one. We want to measure the trade-offs between these three options and how they affect the way requests are distributed as well as the benchmarking framework.

What is the suggested technique's scalability characteristic? The suggested method for generating requests necessitates message exchanges between several BGClients in order to lock and unlock data items and assess the viability of actions. We intend to measure this overhead and how it affects the technique for generating requests' scalability. We should be able to suggest improvements to improve scalability thanks to this intuition. What is the difference between the results achieved using one social graph (the proposed change) and disconnected social graphs (the existing version of BG)? This inquiry relates to systems that might make use of BG's most recent version. To measure any differences, we plan to repeat the published studies, like the ones described in [5].
Our short-term research orientation is shaped by an examination of these questions.

C Closed as opposed to Open

BGClients use a set number of threads to generate requests. Every thread simulates one of the eleven acts carried out by an arbitrary user of a social networking site. The DZipfian distribution is used to condition the member who was chosen at random. Because a thread doesn't imitate a new member until its simulation of a

current member is finished, this model is known as closed emulation. There might be a pause in between each member's replication of an action under this model. In the past, a financial institution with a set number of tellers (ATMs) and concurrent customers (threads) carrying out financial operations concurrently was modeled after this [13].

G Conclusions and Future Research

Last but not least, we are examining the feasibility of a Benchmark Generator, BG+, which takes as inputs an abstraction of an application, its actions and their dependencies, metrics to be measured, and a control attribute. A benchmark unique to that application is its output. Applications having a variety of use cases, like data sciences, should employ this [14]. BG+ is essentially an expandable toolkit that can be set up with its input to let a data scientist quickly create a benchmark for their work. It would reveal how the database schema is built as well as the abstracted operations that the data scientists are supposed to carry out, much like BG and YCSB. As a result, BG+ may support a variety of data models, including unstructured, structured (relational), JSON, extensible, images, audio and video as input.

REFERENCES

[1] X. Bai, F. P. Junqueira, and A. Silberstein. Cache Refreshing for Online Social News Feeds. In CIKM, pages 787–792, 2013.
[2] S. Barahmand and S. Ghandeharizadeh. BG: A Benchmark to Evaluate Interactive Social Networking Actions. CIDR, January 2013.
[3] S. Barahmand and S. Ghandeharizadeh. D-Zipfian: A Decentralized Implementation of Zipfian. In ACM SIGMOD DBTest Workshop, 2013. [4] S. Barahmand and S. Ghandeharizadeh. Expedited Benchmarking of Social Networking Actions with Agile Data Load Techniques. CIKM, 2013.
[5] S. Barahmand, S. Ghandeharizadeh, and J. Yap. A Comparison of Two Physical Data Designs for Interactive Social Networking Actions. CIKM, 2013.
[6] M. W. Blasgen, J. Gray, M. F. Mitoma, and T. G. Price. The Convoy Phenomenon. Operating Systems Review, 13(2):20–25, 1979.
[7] R. Cattell. Scalable SQL and NoSQL Data Stores. SIGMOD Rec., 39:12–27, May 2011. 12
[8] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In Symposium on Opearting Systems Design & Implementation - Volume 6, 2004.
[9] R. Nishtala et. al. Scaling Memcache at Facebook. NSDI, 2013.
[10] A. Floratou, N. Teletria, D. J. DeWitt, J. M. Patel, and D. Zhang. Can the Elephants Handle the NoSQL Onslaught? In VLDB, 2012.
[11] S. Ghandeharizadeh and J. Yap. Gumball: A Race Condition Prevention Technique for Cache Augmented SQL Database Management Systems. In ACM SIGMOD DBSocial Workshop, 2012.
[12] S. Ghandeharizadeh and J. Yap. Cache Augmented Database Management Systems. In ACM SIGMOD DBSocial Workshop, June 2013.
[13] J. Gray and A. Reuter. Transaction Processing: Concepts and Techniques, pages 677–680. Morgan Kaufmann, 1993.
[14] C. Greenberg. Overview of the NIST Data Science Evaluation and Metrology Plans, Data Science Symposium, NIST, March 4-5, 2014.
[15] P. Gupta, N. Zeldovich, and S. Madden. A Trigger-Based Middleware Cache for ORMs. In Middleware, 2011.
[16] D. Patterson. For Better or Worse, Benchmarks Shape a Field. Communications of the ACM, 55, July 2012.