Kafka: A Distributed Messaging System for Log Processing

The paper focuses on log data processing system created for and by linkedin. The paper is a presentation of the system. It describes the needs for the system, the architecture and design, the performance, and discuss future work for the system.

Streams of messages are defined as topics, to which messages are published to. These messages are then stored in server sets; brokers. One can then subscribe to messages from brokers and pull data from them. The system is based on simple efficient implementations.

The system was run in an experimental test comparing results with activeMQ and rabbitMQ. Two tests were run: 'Consumer test' and 'producer test'. The consumer test was run with two different batch configs on kafka (1 and 50), and while no configuration on the other ones (stated that there was no simple way to do this). Both configs gave a much better result compared to rabbitMQ and activeMQ. The rate for batch size of 1 still gave message distribution 2 times higher than rabbitMQ, which was the most efficient of the two "competitors". With batch size of 50 kafka sent on average 400.000 msg/sec, which is miles above all other test results.

The consumer test showed that kafka consumed 4 times more messages than activeMQ and rabbitMQ. The reasons pointed to were efficient storage format, the lack of need to maintain delivery state of every message and that the sendfile API reduces transmission overhead.

The motivation of the paper is to describe the a new distributed messaging system developed by linkedin. The tool has been open sources, so a possible motivation would be to describe and in a sense offer the free system to users.

Cassandra - A Decentralized Structured Storage System

The paper presents Cassandra, a distributed storage system for managing large amounts of structured data spread across different commodity servers, developed by facebook. The paper focuses on describing the data model of the system, the client API, system design, and applications for Facebook specifically.

The results from the paper are stated in a form of experience from practice using the system. This showed some scenarios with both positive and negative sides to it.

The motivation for the paper is to show off the newly developed tool, and show how it can be applied in practice, and give an insight to how it works in correlation to the facebook service.

Workload Analysis of a Large-Scale Key-Value Store

The paper aim to analyse the workload of key-value storage. The research of the paper was done by collecting traces from Facebook's memcached deployment. There were over 284 billion requests from five memcached over several days. The five polls had different characteristics mostly size (number of requests), and types of requests; DELETE, UPDATE, GET.

The paper outputs a lot of data, given the huge data set and the big number of angle the data set was researched. The results overall give a good overview over all the different pools of workload. Most of the results were specific to the type of pool, which give an overview of how different types of pools act.

Some important pointed out points were that for the ETC pool a solution to improve hit rates here would be to increase RAM or servers, but the paper shoved that this would yield a very marginal

improvement. Also that on a general basis that it is important to increase the memcached's hit rate on larger data.

The motivation is pretty clear. The paper states that little is known about key-value workloads. It is an important aspect for scale-out companies, but only companies that actually operate them have the knowledge. So the motivation is to give a analysis/clear picture over how this works.

Review of: Casandra - A Decentralized Structure Storage System

The paper presents Casandra, a distributed storage system developed and used by Facebook. Casandra tries to maintain a high write throughput and no single points of failure.

The following core characteristics were identified as requirements and discussed in the paper: Partitioning, replication, membership, failure handling and scaling.

Partitioning is addressed through Casandras ability to scale incrementally and to partition data over a set of nodes. Data is also replicated over N-number of nodes to maintain a high availability and durability.

Membership in the Casandra cluster is based on Schuttlebutt, and failure detection uses a version of Accrual Failure Detection. Failure detection is then reported as a likelyhood or suspicion that a node is down, rather than as a true/false boolean.

New nodes in Casandra takes over the responsibility of parts of a range from other heavy loaded nodes. Transferring of data to a new node in production environment reaches around 40MB/s. Improvement of this speed needs further research.

Review of: Kafka: a Distributed Messaging System for Log Processing

The paper presents Kafka, a distributed messaging system for log processing developed at LinkedIn. Kafka were built with performance and a high throughput in mind, and the paper compares Kafka with other similar log processing systems.

Clients of Kafka uses pull requests to get data. This way, clients can pull whenever they are ready. Kafka also only promises at least once delivery, reducing the checks needed to deliver a message, and moving the complexity over to clients if those features are needed.

Kafka were implemented and tested at LinkedIn. As discussed in the paper, Kafka were tested against Apache ActiveMQ and RabbitMQ. All on the same server configurations. In overall, Kafka fared much better. Mostly because it did not wait for acknowledgments, and because Kafka were able to work with logs in batches.

Review of: Workload Analysis of a Large-Scale Key-Value Store

The paper analyses the use of Memcached at Facebook, and measures the efficiency of the caching server in a large distributed network.

Memcached on Facebook serves five different 'domains'. USR, APP, ETC, VAR and SYS. The analysis were conducted by analyzing requests to the cache servers using a purpose built software called mcap. Mcap allowed the researchers to capture request data with a low overhead compared to the usage of general purpose tools like TCPDump.

The efficacy of the cache were measured by hit-rate. Among the domains in the cache, ETC were the most 'general purpose' one, and therefore the domain discussed the most in the paper.

Most of the misses in ETC came from 'new hits'. About 50% of the cached elements were only requested 1% of the time. Most of the values were also quite small, with a few larger elements (around 1MB). The paper suggests that removing large objects might improve the overall hit-rate.

Cassandra - A Decentralized Structured Storage

System In this paper they introduced Cassandra (distributed storage system) and built, implemented, and operated this storage system providing scalability, high performance, and wide applicability. Cassandra aims to run on top of an infrastructure of hundreds of nodes, which does not support a full relational data model. This model supports dynamic control over data layout and format. The Cassandra API consists three simple methods (1. Insert (table,key,rowMutation), 2. get(table,key,columnName) , 3. Delete (table,key,columnName)). They demonstrated that Cassandra could support a very high update throughput while delivering low latency.

Workload Analysis of a Large-Scale Key-Value Store

This paper aims to understand the realistic key-value workloads (Key-value stores are a vital component in many scale-out enterprises) outside of the companies. They propose a simple model of the most representative trace to enable the generation of more realistic synthetic workloads by the community. They analyze the workloads from multiple angles, including: request composition, size, and rate; cache efficacy; temporal patterns; and application use cases. This paper presented a dizzying number of views into a very large data set, which tell a coherent story of five different Memcached (ETC, APP, VAR, USR, SYS) workloads at Facebook.

Kafka: a Distributed Messaging System for Log Processing

This paper introduced a novel system called Kafka (a distributed messaging system) for processing huge volume of log data streams with low latency. Kafka achieves much higher throughput than conventional messaging systems because they focused on log processing applications. They have been using Kafka successfully at LinkedIn for both offline and online applications where they incorporate ideas from existing log aggregators and messaging systems.

Cassandra - A Decentralized Structured Storage System

Cassandra is a distributed storage system for managing very large amounts of structured data spread out across many commodity servers, while providing highly available service with no single point of failure. Cassandra aims to run on top of an infrastructure of hundreds of nodes.One of the key design features for Cassandra is the ability to scale incrementally.After the all the experiment they got the conclusion, Cassandra can support a very high update throughput while delivering low latency.

Kafka: a Distributed Messaging System for Log Processing

Kafka is a distributed messaging system that we developed for collecting and delivering high volumes of log data with low latency. their system incorporates ideas from existing log aggregators and messaging systems, and is suitable for both offline and online message consumption. They made quite a few unconventional yet practical design choices in Kafka to make our system efficient and scalable. The experimental results show that Kafka has superior performance when compared to two popular messaging systems. They have been using Kafka in production for some time and it is processing hundreds of gigabytes of new data each day.

Workload Analysis of a Large-Scale Key-Value Store

Key-value stores are a vital component in many scale-out enterprises, including social networks, online retail, and risk analysis. Accordingly, they are receiving increased attention from the research community in an effort to improve their performance, scalability, reliability, cost, and power consumption. To be effective, such efforts require a detailed understanding of realistic key-value workloads. And yet little is known about these workloads outside of the companies that operate them. This paper aims to address this gap. To this end, They collected detailed traces from Facebook¡¯s Memcached deployment, arguably the world's largest. They analyze the workloads from multiple angles, including: request composition, size, and rate; cache efficacy; temporal patterns; and application use cases. They also propose a simple model of the most representative trace to enable the generation of more realistic synthetic workloads by the community. their analysis details many characteristics of the caching workload. It also reveals a number of surprises: a GET/SET ratio of 30:1 that is higher than assumed in the literature; some applications of Memcached behave more like persistent storage than a cache; strong locality metrics, such as keys accessed many millions of times a day, do not always suffice for a high hit rate; and there is still room for efficiency and hit rate improvements in Memcached's implementation.

Cassandra: a decentralized structured storage system

This paper describes a distributed storage system called Cassandra, which is also used by Facebook (eg. saving user messages). The system is highly scalable and thanks to the geographical distribution high latency is no longer an issue. First, the authors describe the data model: every table contains multidimensional rows indexed by keys. Columns can be either simple or structured. Next, they thoroughly inspect the key characteristics of their system. The system uses a consistent hashing that creates a kind of a 'ring': when addressing an item, the hash of the key determines its position on the ring. Each node is responsible for part of the ring and for replicating the items saved there. When a new node appears, it chooses the place where it would alleviate a more heavily loaded node. In order to fulfill the task of data persistence, it stores the data locally. The system also uses a commit log feature and a bloom filter containing information about which keys are in which file.

Workload Analysis of a Large-Scale Key-Value Store

The paper's main goal is to analyze workloads of Memcached, Facebook's key-value store. The authors try to expose the underlying patterns of user behavior. Among the key contributions are 1) determining the hit rate, 2) show of daily and weekly patterns and 3) analytical model. First, they describe Memcached's architecture, its distribution among clusters and the tracing methodology - using their packet sniffer mcap. They also talk about the division of servers into pools, that is in fact a division into different namespaces. Next, they propose many characteristics of the workload (types of requests, patterns, etc.). Authors also thoroughly inspect the problem of determining the hit rates of single pools. In the final part, they present their distribution models.

Kafka A Distributed Messaging System for Log Processing

This paper presents a lightweight messaging system for log processing Kafka, which is used at LinkedIn. The system overcomes alternative solutions when it comes to simple log processing thanks to low delivery guarantee, API, distributed support and persistence of messages. The system is based on the pub/sub concept. A producer publishes messages of varies types (topics) to brokers. Then a consumer can subscribe to one or more topics to get messages from brokers. There were several details that made the system efficient: 1) simple storage: each partition of a topic corresponds to a log, whose offset serves as an address for

the message, 2) efficient transfer: they use prefetching, single caching and sendfile API and 3) stateless broker: the overhead of each broker is reduced (consumers take up several responsibilities). When the distributed side of the system is concerned, the authors mention the concept of consumer groups – the goal is to divide the messages evenly among the consumers. The system uses exclusive access to partitions. The system is decentralized, without a master node and using consensus service Zookeeper. Kafka guarantees at-least-once delivery as the exactly-once type would be unnecessary. Finally the authors present the usage of the system at LinkedIn and propose two experiments, where they compare the speed of consuming and producing of Kafka with two alternative systems. At both cases Kafka shows orders of magnitude better results.

Workload analysis of a large-scale key-value store

Distributed hash table exists for many years and the corresponding Key-Value (KV) store techniques are used by many famous enterprises such as Amazon, Facebook, Twitter and Linkedin. Although there are many research works devoting to improving KV store techniques, few work details the large scale KV-store workload in companies.

This paper tries to bridge the gap between academy area and industry area based on the traces from Facebook's Memcached deployment. It analyzes the request composition, size, rate, cache efficacy, temporal patterns and application use cases. In addition it also propose a model to generate more realistic synthetic workloads.

The experiment is fully detailed and the data set is suitable, including 284 billion requests. Firstly, it analyses the request composition and corresponding ratio. An interesting discovery is that the ratio of GETs to UPDATEs in ETC (general purpose request) is 30 which is much higher than most synthetic workloads assume. Secondly, request sizes and temporal patterns are considered. It shows small values dominate almost all workloads and diurnal cycle exists. Thirdly, it illustrates hit rete changes with the time and pool. Fourthly, it presents most keys do not repeat frequently while the percentage of unique keys out of total in 5-minute bins is up to 74.7%. Finally, it shows the statistical modeling based on key-size distribution, value-size distribution and inter-arrival rate distribution.

There exists two open problems in this paper. One is how to design an optimal cache replace policy. Another one is the whether it is necessary to improve the hit rate as it may cause extra cost in other respects.

This paper details workload in a large-scale key-value store and have many interesting observations. However, it can be researched further, especially in statistical modeling. If possible, it will be very meaningful to investigate the relationship between three distributions.

Cassandra a decentralized structured storage system

With the popularity of commercial storage service, more and more companies are willing to pay for that. However, before enjoying this service, a technique problem that has to be solved is how to manage the data over thousands of servers.

This paper proposes a distributed storage system, Cassandra, which supports managing very large amounts of structured data across many commodity servers without single point failure. The feature of this system is able to manage the persistent state in facing nodes failure and provide dynamic control over data layout and format.

The paper details the system architecture, including partition, membership, bootstrapping, scaling the cluster and local persistence. Consistent hashing is used in partition and replication is made to achieve high availability and durability. In failure detection, a value is used to represent the possibility of

failure instead of a Boolean value. Kernel copy technique is used in migrating from one node to another node.

The paper also gives the performance analysis. It shows the median time is 15.69ms in search interactions and 18.27 in term search under 50+TB data on a 150 node cluster.

Kafka: a distributed messaging system for log processing

Log is the diary of the program activities. With the information age coming, the volume of log is experiencing exponential growth. Log processing is becoming a new challenge.

This paper introduces a distributed messaging system, Kafka, which can collect and deliver high volume of log data with low latency. For efficiency by partition, three changes are made. In Kafka, each partition of a topic corresponds to a logical log and a physical log is implemented with a set of same size files. In addition, explicitly caching messages in memory is avoided. Last change is that the information about how much each consumer has consumed is not kept by the broker, but by the consumer itself. For distribution, two choices are made. One choice is that all messages from one partition only can be consumed by one consumer within each consumer group. Another one is to not have central master node.

The paper also presents the performance analysis. It compares the experiment results with ActiveMQ and RabbitMQ and shows at least two times higher than both of them in producing messages and more than four times in consuming messages.

This paper also introduces the future work areas. First, it is to add built-in replication of messages across multiple brokers. Second, stream processing functions are added to this system.

Comments for both papers

Both Kafka and Cassandra are decentralized system while Kafka aims at log processing and Cassandra focus on structured data storage. Both systems need to adopt partition to realize scalability and redundancy to guarantee availability. Although both papers have performance analysis, it is not enough for readers to understand the characters of systems. If possible, a figure had better be used to show the system architecture of Cassandra so that it is more readable.

Cassandra -- A Decentralized Structured Storage System

Cassandra is used by Facebook to manage its huge amount of structured data in a distributed manner. Cassandra aims to run on hundreds of cheap commodity hardware to provide a simple data model that supports dynamic control over data layout and format.

Cassandra partitions data across the cluster using consistent hashing. Each node in the Cassandra cluster obtains a position on a ring generated by consistent hashing. Each data item is stored on a node on the ring. To find the node, Cassandra hashes the item’s key to yield its position on the ring. Then the algorithm searches the ring clockwise to find the first Cassandra node whose position is greater than the item’s position. This node is the coordinator of this item and the item is stored on its coordinator.

Cassandra uses replication to achieve high availability and durability. The coordinator of a data item is responsible for the replication of all the items within its range. Cassandra allows applications to choose replication schemes ranging from “rack aware” and “rack unaware” to “datacenter aware”.

Cassandra uses a modified version of the phi Accrual Failure Detector to detect if a node is up or down. Phi Accrual Failure Detector emits a value which represents a suspicion level for each of monitored nodes. The greater the phi is, the more accurate the result is.

When a new node joins Cassandra, it obtains a random token as its position on the ring. It's position information is propagated to the leader of the cluster as well as other nodes in the cluster using gossiping. Data items falling within the range of the new node are moved to the node using kernel to kernel copy.

Data are first written into a commit log. After successful update of commit log, the data is written into an in memory data structure. When the in memory data structure exceeds a certain threshold, it is dumped to disk. When there is a read, Cassandra first looks it up in its in memory data structure, if not found, it checks its bloom filter. It only loads data back to memory when the bloom filter indicates a possession of the data.

Cassandra was tested on 100 nodes across the US. All Facebook messages were copied to Cassandra to test the application of Inbox search. Experiment results show that Cassandra can support a very high update throughput while delivering low latency.

Workload Analysis of a Large-Scale Key-Value Store

This paper analyzes a Facebook's large-scale key-value store -- memcached. Memcached is a distributed hash table that only stores data items in memory. Clients use consistent hashing to select a unique server for a key or item. Memory is divided into slabs. Unused memory is managed in a heap, used memory is managed by slab classes. A data item first searches slab classes for memory, if not found, it requests a new slab from the heap. LRU is used to evict data items out of memory when memory is full.

Memcached servers are divided into pools with each pool storing data generated by different type of applications.. In this paper, five pools are analyzed: USR, SYS, APP, ECT, and VAR. The authors use a self-developed packet sniffer called mcap to capture packets sent to and from memcached servers. The captured traces vary from 3TB to 7TB.

Experiment results show that the number of GETs sent is much greater than that of UPDATEs in ETC (30:1). Over 90% of APP's keys are 31 bytes long, and values sizes around 270b show up in more than 30% of SET requests. Traffic in ETC bottoms out around 8:00 and has two peaks due to the time difference between north america and europe. The hit rate of most of the pools are above 90%, except ETC which is 81.4%. 50% of keys do not repeat many times, with a few keys repeat millions of times. Count of reuse of keys decay quickly after the first hour. Case study results show that key size, value size, and inter-arrival rates are independent of each other.

Kafka: a Distributed Messaging System for Log Processing

Kafka is a distributed messaging system for log exchanging and processing. It combines the pub/sub paradigm and point to point delivery. Brokers are used to route messages from log servers to log consumers. Messages or event logs are classified into topics. Each topic is further divided into partitions to reduce mutex. A consumer can subscribe to one or more topics. A producer accumulates its log events, when the size reaches a certain amount, it publishes an event to one of the brokers. The broker appends the received log events into its segment file. If it has received enough logs or the time has elapsed long enough, it notifies the consumers. A consumer then pulls the latest log events from the broker. Any log event is kept in a broker for only 7 days, after that, it is deleted. A zookeeper is selected to manage all the brokers and consumer groups. Kafka guarantees at-least-once delivery.

Experiment results show that Kafka outperforms pure pub/sub messaging systems -- ActiveMQ and RabbitMQ in terms of producer and consumer message throughput.