Energy Efficient Key/Value Store

Short Version of The Dissertation

Frezewd Lemma Tena
Technischen Universität Dresden
Fakultät Informatik

July 10, 2017

1

Introduction

Energy conservation is a major concern in today's data centers, which are the 21st century data processing factories, and where large and complex software systems such as distributed data management stores run and serve billions of users. The two main drivers of this major concern are the pollution impact data centers have on the environment due to their waste heat, and the expensive cost data centers incur due to their enormous energy demand. If we look at the total cost of ownership(TCO) of a data center, energy cost accounts for over 35% of the TCO. As early as 2005, Google has warned that how power costs can easily surpass server costs by a large margin if the power consumption trend in the data center continued business as usual. On the other hand, if we see the current state of the industry, the industries energy efficiency for most data centers is not more than 50%. Thus, companies continually seek energy-efficient hardware and software system components to curb the cost of running a data center.

In this thesis, we present a noble solution to address one of the main source of energy consumption in the data center—the storage subsystem. Different researchers put the storage subsystem contribution between 26% and 40% of the total energy consumption of the data center. Many researchers have attempted to address the energy consumption problem of the storage subsystem at both the software and hardware levels. We contribute to the domain of the software solutions at the distributed level by creating an energy efficient, distributed key/value data store—a data management system that stores and retrieves data as a simple key/value pairs, which basically make them a distributed hash table.

Key/value stores are popular data stores in data centers. These stores, typically, run on multiple, distributed nodes across multiple data centers in a geographically wide area setup. Thus, in key/value stores, data is distributed and replicated a number of times to immensely improve the systems performance, availability, and scalability. However, availing the same data on a number of nodes in different data centers and letting these nodes run 24/7 at all utilization levels reduces the energy efficiency of the storage subsystem and hence significantly increases the total cost of ownership of the data center. Therefore, in our re- search we ask the following questions: *how to create a key/value data store that distributes and replicates data as usual but reduces its energy consumption when it is under a state of low utilization? how can we improve the performance of the store that we may loose due to the introduction of the energy saving mech-*

anisms? how do we make the key/value store adapts itself to be more energy efficient when there is an opportunity to use a cheap source of energy in the system?

It is not in this thesis's scope to answer the research questions in all kinds of key/value stores or other types of distributed data management systems in a data center. In our work, we solely focus on introducing energy saving mechanism in key/value stores that are based on consistent hashing algorithm for their data distribution and has a master-less, decentralized approach for client requests coordination. All of our algorithms are implemented in the open source Apache Cassandra to create an energy invariant of it called PowerCass.

This thesis makes the following contributions:

- The first consistent hash based, energy efficient distributed key/- value store.
- The techniques to dynamically place data in a consistent hash based system without changing the system's fast look up of keys.
- A data and applications co-locating mechanisms which is driven by the energy efficient data store's perspective of the system utilization.

2

Background

Key/Value stores mainly serve in data centers. Any solution towards improving the energy efficiency of a key/value store should be able to integrate with existing solutions for data center energy efficiency. This is because any new energy saving solution for data centers may end up in one of two possibilities: either it contributes to the energy saving capacity of a data center or it creates interference with the existing solutions. In our work, we have identified state of the art, and put our work in context by reviewing and classifying existing solutions in the literature into four groups: facilities level, components level, node level and group level solutions.

The *facilities level solutions* are those solutions which can not be categorized as component, node or group level. They are related to data center power, cooling, and network cabling infrastructures;

The *components level solutions* are solutions related to the main components of a data center server: CPU, memory, disks, and network cards.

The *node level solutions* limit the power consumption of an entire node with minimal or no performance degradation; These solutions require a mechanism

to *intelligently manage all the component subsystems together*, not just a single component.

Solutions at the group level take into consideration the states of all the nodes in the group rather than the state of an individual node. Therefore, there is a need to collect and process information about each and every member of the group. Our work mainly belongs to this level. Even though there exist several energy saving works related to distributed file systems and data management stores, to the best of our knowledge we are the first to address power saving *in consistent hash based key/value store* through turning off nodes during low activity periods.

3

PowerCass: Energy Efficient Key/Value Store

As we already indicated above, we are not the first to investigate the storage stack to increase its energy efficiency. However, to the best of our knowledge, we are the first to consider powering off entire storage nodes to reduce the consumption of a consistent hashing based key-value stores. Consistent hashing is used in popular data stores like Amazon Dynamo and Apache Cassandra. We have based our study of creating an energy efficient, consistent hash based key/value store, PowerCass, on Apache Cassandra which is in turn a descendant of Amazon Dynamo.

Neither the original Amazon Dynamo, nor the open-source clone Apache Cassandra addresses the issue of saving energy during low activity times. However some of the techniques Cassandra uses to improve its performance indirectly help saving energy. Such techniques includes the use of bloom filters, caching rows and keys in memories, and sequential writing to the disks. All these techniques help reduce different disk activities. Many studies, as is indicated in the background section, show that techniques that reduce disk activities has the potential to reduce energy consumption of a node. However, how much these techniques save energy is not in the scope of this work and left for future work. However, as we see in the evaluation section, their effect is minimum compared to powering of a Cassandra node entirely as the idle consumption is the biggest source of energy waste.

So, our goal is to introduce the idea of temporarily switching off nodes into Apache Cassandra to create an energy efficient store. Creating PowerCass on top of Cassandra demands us adapting the implementation of Cassandra in such a way that it allows nodes to be shutdown but not considered as failed.

Our main assumption is that powering off storage nodes is possible in Internet scale services because workloads are known to have diurnal patterns, and resources are idle or have a low utilization for the non-peak activity times. In this work, we want to shutoff nodes in tiers following the daily usage pattern, instead of arbitrarily switching off individual nodes. We have three tiers, each consisting of 1/3rd of the nodes. The nodes are classified by their activity pattern, i.e., how long they are on- or offline. We call them active, dormant, and sleepy. While active nodes are constantly on, nodes in the other two tiers are powered off from time to time. The distinction between dormant and sleepy nodes is that the latter only are powered on very infrequently, i.e., they sleep most of the time. We only power them on either to synchronize their state with other replicas or during peak workload activity.

With our three tier division of storage nodes, we can potentially power off 2/3rds of the nodes, saving up to 66% of energy, com- pared to the always-on baseline configuration. When adapting Cassandra to support the notion of powered off nodes, many aspects of the distributed protocols underlying it must be considered. Concepts such as hinted handoff and always writable property must be maintained. We have used different techniques to achieve the maintaining of all the existing desirable feature of Cassandra: re-arranging nodes on the consistent hashing to support availability, power-oriented replication strategy, distributed logging, and promotions for fault tolerance,

In short, we proposed an energy efficient key-value store based on consistent hashing. We partition the storage nodes into three classes, active, dormant, and sleepy. The nodes in the dormant and sleepy tiers are powered off at times of low workload activity. Powering off the nodes has no effect on data availability, i.e., all the data is still accessible. We intend to store replicas in different data centers to improve durability and fault tolerance in the face of failures. We have to adapt the data placement strategy within the original Cassandra to ensure this property.

we have evaluated PowerCass using the well know Yahoo Cloud Serving Benchmark (YCSB), which provides six types of Internet oriented workloads. In all workload types, PowerCass proved to save energy up to 66% during low activity times.

4

Dynamic PowerCass

Even though PowerCass is able to save tremendous energy with the approach of switching off groups of nodes in tiers during low activity time, it suffers from performance degradation and unavailability from powering off nodes. This per-

formance degradation further exacerbated by the static and random placement nature of consistent hashing. Because, random and static data placement strategy may result in a sub-optimal placement of data. Thus, our goal, in this chapter, is to improve the performance of PowerCass through a dynamic data placement strategy that replicates data where it is frequently accessed.

In contrast to clusters that place and lookup data using a central- server(meta-data server) approach, consistent hashed based clusters enable each node to place and lookup data by itself. As a result, in contrast to the central approaches, consistent hash based systems do not suffer from a round-trip delay in order to locate the data. However, consistent hashing, like all other random placement approaches, is unconcerned to locality of data, meaning it does not address the issue of placing replicas on nodes in data centers where data is frequently accessed by the clients. Thus, systems such as PowerCass faces the age-old problem of data placement optimization among the nodes of the cluster.

Data placement optimization can be addressed in two ways: the first and the most traditional approach is to formulate and solve it as an integer linear programming, and the second approach is to solve it by minimizing the number of variables in a repeated rounds based optimization. In this work, following the second approach, we show an intelligent, dynamic data placement strategy which maintains the energy saving capability of PowerCass while maximizing data access locality that enhance PowerCass's performance and availability.

In order to efficiently and automatically relocate objects, the dynamic placement algorithm needs to address three challenges: first, it should be able to trace the access patterns of an enormous number of keys in the system; second, it should be able to keep the fast lookup property of the existing consistent hashing mechanism; and finally, it should consider the available maximum storage capacity of nodes when dynamically replicate and move objects across the data centers.

To address the first challenge, which is to be able to trace large number of keys, we adopt a stream analysis algorithm studied in many research papers. To address the second challenge, we keep a small, distributed mapping information that stores the current owner of a relocated key. To address the third and final challenge, we rely on the assumption that storage is becoming cheaper and can be addressed by using the easy, horizontal scalability property of the store to quickly accommodate an arising needs.

Accordingly, we created an algorithm to introduce an energy efficient dynamic data placement strategy in PowerCass. The most important characteristic of the algorithm is to maximize performance while keeping the energy saving capability achieved through powering off of groups of nodes. The algorithm basically conducts the following steps:

1. Information Collection: Each node gathers information about keys which are not handled locally by the node.

2. Identify top-k keys: While gathering information on remote keys, the node identifies the top-k keys that generate the most remote operations among the stream of keys, using an adapted top-k algorithm (discussed below). Once the top keys are identified, the node determines whether the nodes responsible for the identified top-keys exist in the same data center with itself and/or whether they are already relocated keys. If the responsible nodes exist in the same data center or the keys are already moved, those

keys will be removed from the selection list and the next candidate key (top-k+1) is promoted to the top-k list.

3. Send top-k Information: When PowerCass enters to an epoch where it requires to move hot keys around, each node sends information about the top-k remote keys it identified to the active node that is considered the primary replica for the key according to consistent hashing. This means, the algorithm will not send the keys to dormant and sleepy nodes. From this step onward and until the last step is over, all nodes cease gathering information on top-k keys and only process the information on already identified keys.

4. Identify the top 5% and decide best place: Every active node that received its top-k remote keys sorts and selects the top 5% hot keys among all the top-keys identified by the sender nodes. Then for each chosen key, the algorithm decides the best location (data center) based on the frequency of the key requested at each sender node. Before disseminating the information, PowerCass check that whether there are replica nodes for the keys in that best data center. Because, it may be better to move the keys from the replica in the same data center than to move them from a remote data center over a wide area network. This is highly useful specially when the time approaches to wake the sleeping nodes. If the algorithm decides to move the key from the node in the same data center as the key's destination node, PowerCass wakes up the sleeping node at this step.

5. Move the keys: Once the best place for the keys has been determined, Powercass move the keys to the new place. We use a direct communication instead of gossiping to transfer the keys between the two nodes.

6. Disseminate the new location information: Finally, each node disseminates the new location of the keys using gossiping. When a node gets the gossip information, it updates its local relocated keys map.

We have evaluated the algorithm using YCSB and found that the algorithm improved the performance of the store while still keeping its energy savings. Specifically, there is 11% improvement for read mostly workload, 14% improvement

for 100% read only workload, and 10% improvement for read latest workload.

5

Application and Data Co-location Using PowerCass

In this chapter, we use PowerCass to schedule applications and data together and further improve its energy efficiency when it is running on a system of small data centers called micro-clouds.

The small data centers (micro-clouds), which we are considering in this work, are similar to the Microsoft's data furnace in that the waste heat from the computing infrastructure is reused in the surrounding building. So, our main goal is to co-locate applications with their data in the same micro-cloud that currently has enough capacity and provide the cheapest energy cost due to waste heat usage.

The main technique we used is a Data Store Driven Scheduling (DsDs) approach. This approach considers PowerCass's strategy to save energy, PowerCass's perspective on the current workload state of the system, and waste heat demand of the surrounding building of each micro-cloud. In short, we combine PowerCass's power saving strategy with the heat reuse opportunity in the micro-clouds to reduce cost for the micro-cloud provider through more energy saving. Clearly, doing so requires to maximize the utilization of those micro-clouds that currently have heat demand from the surrounding building. In other words, we need to move computation away from those micro-clouds that currently have no heat demand, but still be able to compute where the application data exists—hence co-locating applications and data.

We show that the co-location approach is suitable for addressing not only energy efficiency, but also it addresses performance issues in a distributed micro-cloud system setting. For example, in regard to energy: the approach guarantees more saving by making sure that computing occurs in a data center where energy is cheap due to the reduction of cooling cost; the reduction of the cooling cost comes from the fact that waste heat is being reused by the surrounding building. In regard to performance issue: as the approach allows computing to occur where data exists, it saves the system from moving big data across micro-clouds in a wide area setup. Moving big data across data centers is known to negatively impact the performance of a distributed system.

Saving energy by selling the waste heat requires to produce enough heat to meet all the demands at all the sites. This requirement calls for a scheduling approach to co-locate computing and data access jobs on a micro-cloud or micro-

clouds where waste heat is in demand. The scheduler also needs to address the following related, list of issues.

First, a naive co-locating of applications and data at a site for example, by only considering energy requirement may have the obvious consequence of overloading the selected site beyond its capacity. Therefore the scheduler needs to take the issue of load and energy balancing in to consideration.

Second, in a distributed system spread over a wide geographic area, geographically induced latency makes moving large size data across data centers highly cost and performance ineffective. Thus, avoiding large data movement is important for the the scheduler.

Third, the scheduler needs to avoid frequent oscillations of appli- cations among micro-clouds: frequent oscillation leaves little time for the applications to make an acceptable progress.

Fourth, the scheduler needs to deal with long and short tasks differently. For example, short and interactive jobs are highly latency sensitive than long and batch oriented tasks. Short tasks demand immediate scheduling and execution, whereas long tasks demand ample resources to meet their performance goal. It is highly challenging to satisfy the requirements of both long running tasks in the existence of short lived ones.

Fifth, the scheduler needs to deal with priority of tasks to fairly discriminate among new tasks competing for resources, and those tasks are failed needs to be rescheduled.

Finally, the scheduler needs to seamlessly work with the underlying energy efficient data store.

We have created a scheduler algorithm which conducts the following activities to achieve the above requirements in a system of micro-clouds:

1. At each active node, DsDs continually collects read and write data access metrics. Then at each minute it communicates the metrics information to its peers.

2. At each micro-cloud, a designated node continually receives energy consumption metrics from the building heating system and later on distribute the information to all the non-designated nodes that are in the same micro-cloud. Then, at each minute, the designated node sends the energy information to the rest of the micro-clouds as well as receiving their information too.

3. When applications put a data access request, DsDs look for the three nodes responsible to handle the requested task using the data store's consistent hashing algorithm. From this information DsDs drives the three candidate sites for the application placement.

4. Once an application execution site is chosen, DsDs proceeds with start/stopping the application. If the application is an already running application in a different cloud, DsDs stops it in the current cloud and starts it in the new cloud. But if the node in the destination site is a dormant or sleepy node, DsDs makes sure that the node is waked up, synchronized and promoted to the active status before starting the application; and also do the reverse to demote the active node to a dormant or sleepy status in the current micro-cloud.

5. When subsequent requests arrive after the application is started in its new site and before the next scheduling period is reached, DsDs redirects all the requests to the new destination without any scheduling delay.

We have evaluated the algorithm using a simulation and found out that it is able to improve the energy saving capability of PowerCass at least by 20% which push the total saving more than 85%

6

Conclusion

In this thesis, our focus was to develop an energy efficient key/value store. In particular, a consistent hash based, distributed, highly available, and fault tolerant data management system. The primary research questions were :

- how to create a key/value data store that distributes and replicates data as usual but reduces its energy consumption when it is under a state of low utilization?
- how can we improve the performance of the store that we may loose due to introduction of the the energy efficiency mechanisms?
- how do we make the key/value store adapts it self when there is an opportunity to use a cheap source of energy in the system?

We have shown in Chapter 3 that how to use powering down idle nodes during low activity time and addressing arising issues like write availability, fault tolerance, and load balancing answers our first question. In chapter 4, we have shown that dynamically placing a small subset of popular data, where it is needed most, answers our second question. The dynamic placement technique managed to regain the performance lost due to powering off nodes. Finally in Chapter 5, we combined the energy efficiency mechanism(intelligently powering off/on nodes in a group) with data and applications co-location with data store driven scheduling as a way of adapting the store to use cheap energy sources.

In the future, we would like to redesign PowerCass so that it is able to use an integrated power saving techniques from the rest of the stack with the ability of avoiding conflicting techniques.

Our main outlook for the future is that, until the world manages to provide energy from a renewable sources for free, conservation of energy will continue to be the main factor to determine the total cost of ownership of data centers and the well being of the environment.