Using Software Simulation in Green Data Center Design

Hao Jiang, Wenjin Hu, Long Zhang, Vinay Soni, Jeanna Matthews

Department of Computer Science, Clarkson University, Potsdam, NY, United States

April 21, 2013

Abstract

Instead of maintaining large centralized data centers that rely on grid power, using renewable energy or green power to build smaller, distributed data centers can be more competitive in a number of ways such as reducing the transmission costs of electricity and taking advantage of power sources in remote locations. However, the unpredictability of green power introduces substantial challenges to both system design and management. GDCSimulator is a software platform that enables experimentation with the placement and management of green datacenters without the prohibitively high cost of actual data center deployment. In this study, we use GDCSimulator to explore the impact of datacenter placement, models for the availability of solar power and a variety of data consistency models such as eventual consistency, strong consistency and other intermediate ones.

1 Introduction

In this era of cloud computing, companies are building larger and larger data centers to serve the computing and data storage needs of their customers. One of the most challenging and costly aspects of data center management is managing the massive power and cooling requirements. Thus, renewable energy, or green power is becoming more and more important to datacenter placement and management. Google's Dalles Data Center, which receives power directly from a nearby hydroelectric power station[2], is a well-known example.

However, in most of these "green" data centers, green power is only a supplement to traditional grid power. Due to the innate instability and unpredictability of green power, system designers tend not to consider it as the primary power source of distributed data centers. Large data centers still need to be located in some place that has grid power supply. This greatly limits the usage of green power in data centers.

In the GreenDataCenter(GDC) project, we try to solve this dilemma from another direction. Instead of deploying one single giant data center that relies on grid power, multiple smaller data centers that fully rely on green power are deployed instead. These data centers receive power from wind turbines, solar panels and/or other renewal sources.In fact, multiple renewal sources could be combined at one site to complement each other. When there's not enough power, they will simply shut down. By properly arranging these small data center nodes, the goal is to guarantee a minimum level of data center availability at any time and thus provide a continuous working virtual data center to the end user. Without the limitation of power supply from grid, these "pure green" data centers also have a wider range to choose their location of deployment, which means they have a better chance to utilize more green power. It is cheaper and much simpler from a regulatory perspective to bring only a fiber optic network connection to a remote site than to bring electrical transmission capability.

Such a distributed system also introduces big challenge to the consistency model employed by the system. In a traditional data center system, power failure or full data center failure is rare and typically accidental, unexpected and hopefully of short duration. Using temporary secondary power supply can cover many of the anticipated causes of such down time. However, the situation is quite different when switching the power supply from grid to green power. Power failure will become frequent, more predictable and may last for a longer period of time. Some types of green power, like solar radiation and hydroelectric power, are easier to predict. Others like wind power are less predictable. The mechanisms employed for data consistency and availability must be carefully designed to meet the needs of such a system.

In this process, software simulation of green data centers can be a great help. By gathering data such as solar radiation level and wind profile and inputting them to the simulator, the service level, such as failure rate and availability could be evaluated. We can experiment with the impacts of varying parameters for network bandwidth, latency, data center size, datacenter placement, client placement and workload. With GDCSimulator, one can easily develop and verify ideas about distributed systems

without the huge expense of actually citing real data centers.

The rest of the paper is organized as following: The next section gives several cases we have worked on using the simulator, which provide helpful suggestions to the design of future systems. Some background about the GDC project and GDCSimulator as well as important parameters that are critical to simulation result are also include. We also have some discussion about simulating different consistency models. Finally, the paper ends with a conclusive remark and discussion to future work.

2 Case Study

In this section, we describe two cases we have done using the GDCSimulator, including background information, test scenario design and result evaluation.

2.1 Solar Powered Data Center

2.1.1 Background

In our GDC projects, we focus on using solar power to support the green data centers.

Solar power is one of the important kind of natural power that can be used to build green data centers. Comparing to other renewable energy, the solar power is more stable, more predictable, and less expensive. But there also are disadvantages that cannot be overlooked. Of course the clear disadvantage with solar is the loss of power overnight. Without battery set of large capacity, a single solar station is infeasible to serve as a consistent power source.

Under some unrealistically stringent assumptions (e.g. that each site has a radiation of 8 hours daily, that each data center has a complete copy of all data being served and that there is no need to synchronize the data between data centers during the handoff), we could estimate the need for 3 sites. However using our simulator, we can investigate the impact of more realistic solar power models, a wide range of data consistency models, different availability requirements (e.g. requiring N copies of data to be written) and the different requirements for client placement and latency.

2.1.2 Scenario Design

We choose five locations to place our data center, which are desert areas near to the equator. Table 1 shows the name, latitude and longitude of each location. Figure 1 shows the positions on Google Map. We then simulate clients originated from different locations, trying to answer the following questions:

Name	Location
Egypt	(24.69,27.42
Tibet	(38.75,82.88)
Oceania	(-22.92, 127.62)
United States	(34.02,-115.66)
Brazil	(-5.27,-39.90)

Table 1: Locations of Solar Powered Data Center

Distance	Latency (10 ms)
d < 1000km	5
1000km < d < 4000km	20
4000km < d	$100 + \frac{d}{6000km} * 50$

Table 2: Distance and network latency evaluation

- Are these servers enough to build a 24-hour data center system?
- How does the access latency change according to time?
- Does this system have any preference of the client's location?

In the simulation, several important parameters are taken into account.

1. Simulation time period. We decide to simulate one virtual day as the objective of the simulation is to check whether the system could provide 24 hours continuous service. Taking the data magnitude into account, we chose to run the simulator for 8,640,000 cycles, with each cycle representing 10 ms.

2. Network latency. Network latency was simulated by calculating the distance between locations. With latitudes and longitudes, it is easy to calculate the distance between any two points on earth. By performing ping operations to servers located in different countries, we also know that within the same country, network latency is around 10-50 ms, and between continents it is generally over 500 ms. Table 2 shows how we estimate the network latency in the simulation.

3. Sunlight radiation strength. Without loss of generality, we assume that the radiation strength can be described by triangular function, which reach the maximal value at noon. and reach 0 at 4 hours before/after noon. The processing power of data center is proportional to radiation strength. The following equation is used to calculate sunlight radiation

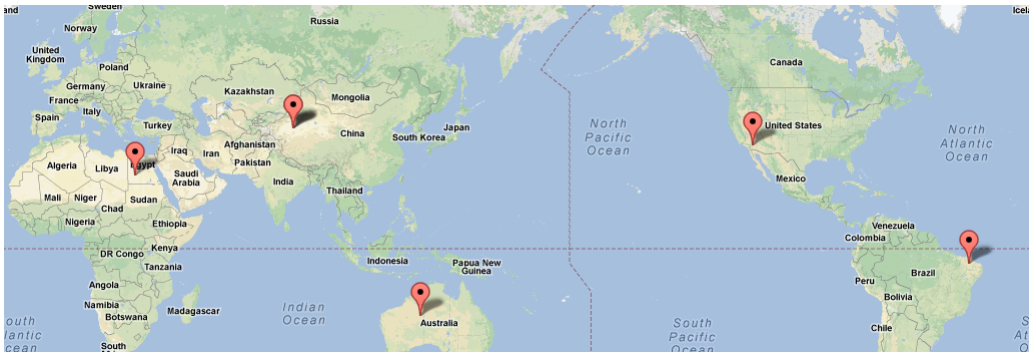

Figure 1: Solar Station Position

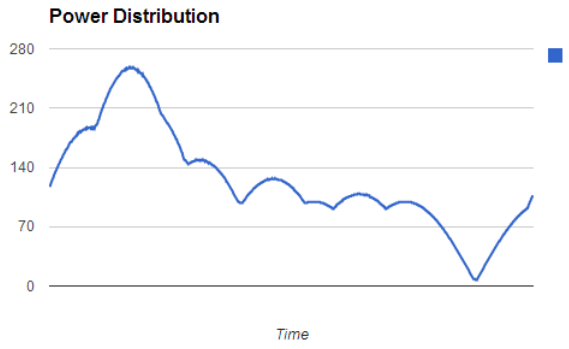

Figure 2: Solar Station Power Distribution

strength.

$$P = B * cos(\frac{(L * 240 + T - 43200)\pi}{28800})$$

where P is Processing Power, B is Base Power, L is latitude and T is current time in seconds

2.1.3 Simulation Result

Our first test tries to evaluate that with these 5 data centers, whether the system can provide 24-hour service. Figure 2 shows how the total processing power varies within a day. From the figure it can be seen that in most of the time, the processing power is over 100%, which means most of the time we have a equivalence of one data center available. However, near the midnight the processing power drop to near 0, then resumes. This is caused by the large gap between the US station and the Australian one, which are separated by Pacific Ocean. Adding one more data center between them may solve the problem. However, islands like Hawaii are mostly covered by tropical forest, which makes them not good choice for deploying a solar power station.

Our next test focus on testing client side latency when accessing the system. Three points from

Name	Position
Montreal,Canada	(45.6, -73.7)
Beijing, China	(39.93, 116.46)
Reykjavik, Iceland	(64.14, -21.87)

Table 3: Choice of client location

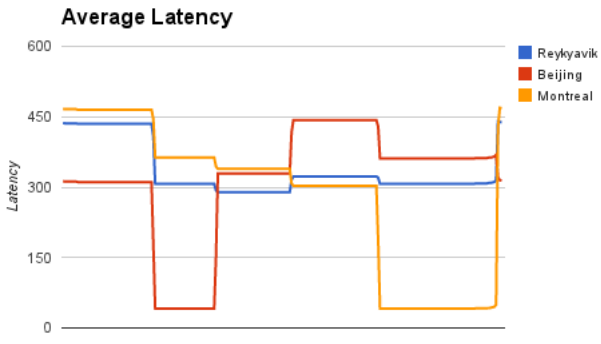

Figure 3: Network latency from different clients

China, Canada, and Iceland are chosen. The first two are used to evaluate latency from a location that is close to one of the data center, which relatively far from the others. The third point, which located in Reykjavik, Iceland, is used to evaluate the performance from a place that is far from any of the data centers, which hopefully represents the worst case scenario.

From Figure 3, it can be seen that the network latency from Reykjavik is always between 300 and 400. For Beijing and Montreal, when using the data center that is close to them, the latency drops to 20-30. However, the latency increase dramatically when switching to other data centers.

In order to minimize the latency, we need to do some adjustment to the data center selection strategy. By default, the strategy is to choose the data center that has most remaining power, without taking the latency into account. We modify the strategy, giv-

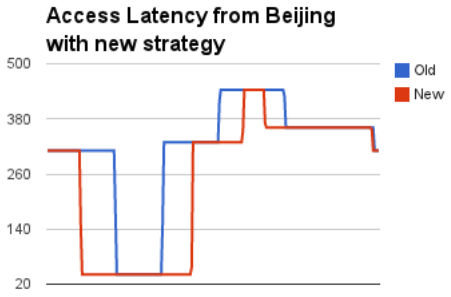

Figure 4: Strategy that minimize the network latency

ing more weight to network latency. Figure 4 shows the access latency from these three cities, after the change.

For clients in all three cities, average latency drops, but clients in Montreal benefit most because they are close to the U.S. data center and can enjoy a longest hours of fast access. After changing the strategy, the average latency dropped from 204 to 122, with a nearly 40% increase. For Reykjavik, the data is 328 to 314, and for Montreal, it is 140 to 102. It can also be seen from the graph that with the new strategy, clients tend to stay as long as possible in the servers with a low latency and keep away from the slow servers.

2.2 Consistency Model

The data consistency model is the heart of distributed system. It describes how different nodes in the system interact with each other. In [1], the authors declare the CAP theorem, which states that a distributed system cannot ensure Consistency, Availability and Network Partition Tolerance at the same time. This means if a system want a stronger consistency constraint, it must choose one from the other two to forsake. One basic requirement to the GDC system is network partition tolerance and it must also maintain high availability. So what we are looking for is a consistency model that sacrifices part of consistency to gain better performance.

2.2.1 Background

The paper [3] describe a popular classification of consistency models. They classify different models based on when the effects of a write operations can be seen by read operations at different sites. The strongest one, which is called "Strong Consistency" or "Linear Consistency", requires that the effects of a write operation will be immediately visible to any subsequent read operation anywhere in the system. The widely adopted two-phase commit protocol is an example of strong consistency. In the contrast, eventual consistency is a weak consistency model which has no specific constraint on the time required before one write operation can be seen on the other nodes. The only requirement is that they will "eventually" become consistent, and that's why it is called "Eventual Consistency".

In [4], the authors describe a "continuous" consistency model that is named TACT. The model introduces 3 parameters: numerical error, order error and staleness. Numerical error defines the maximal number of write operation a node can receive and cache before synchronizing them to others. Order error requires that if a client see some amount of change from other nodes, it need to do synchronization. Staleness controls the minimal synchronization interval. By adjusting these parameters, user can adjust the frequency that nodes do communication with each other, thus gain an acceptable consistency level. Thus the TACT model represents a range of consistency models based on a set of parameters specifying bounds on the level of different kinds of inconsistency or error.

2.2.2 Scenario Design

We choose to first simulate the strong and eventual consistency models, which will be used as the basis of performance comparison. We then simulate the TACT model, adjusting its parameters to alter the performance, allowing us to explore interesting points on the range between strong and weak consistency.

With the prerequisite that the written data must be consistent at least on one node, we focus primarily on the average message latency(time between sending the message and receiving the feedback) and throughput(message count).

In this simulation, we still set the simulation time period to be one day, which is enough to evaluate the latency. As the test involve large amount of client nodes(over 500), too many loops will make the simulation process become very slow. We let one tick represent 100 ms in real world, which means in the simulation of 1 day, 864,000 cycles will be executed.

Network latency between client and server is not as important in this scenario. However, as higher consistency level will introduce more communication between client and server, the network latency between servers becomes more important. In the interest of space, for the graphs presented here, we assume the

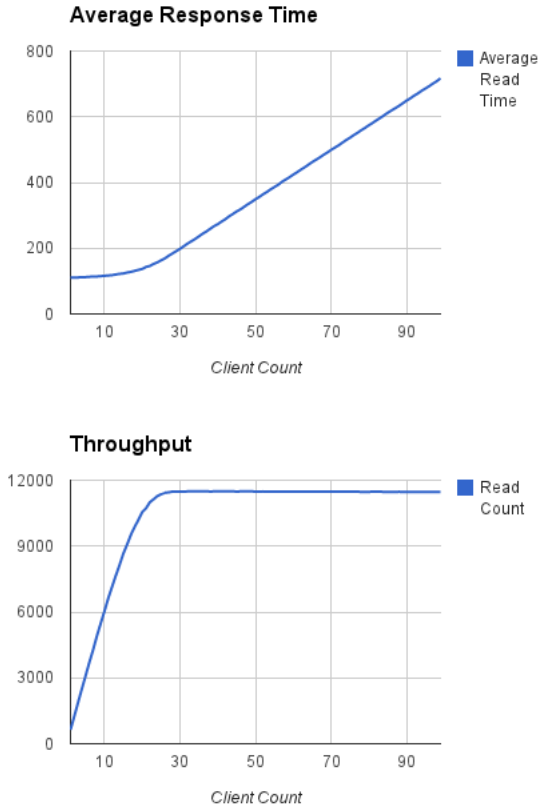

Figure 5: Single Server with multiple Clients

network latency between any of the server pairs is a constant value, but we have explored the impact of increasing server latency as well.

All the scenarios simulating multiple clients send requests to server clusters. By varies the parameters of client/server count, read/write operations percentage, we try to figure out their impact to the system. When doing comparison between different models, we will keep the network latency and processing power unchanged.

2.2.3 Simulation Result

Our first test uses multiple clients sending request to a single server, trying to figure out its capacity. The client count varies from 1 to 100. Each round lasts for one virtual day. Figure 5 shows the result.

There should be no surprise to see that the read latency increase with the client count increase. Another thing we can notice is that the throughput does not increase substantially when the client count exceeds 30. This value varies with the server processing power and time used to process each message.

The second test fixes the client number to be 500,

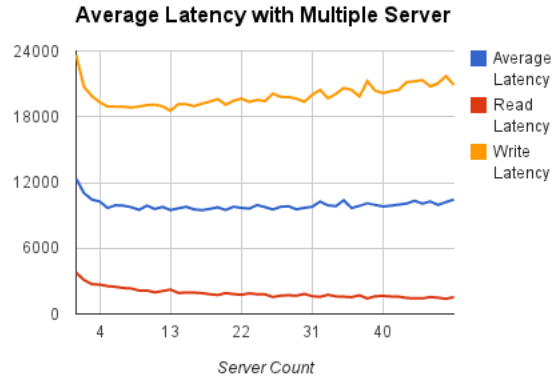

Figure 6: 500 Client with multiple Servers, 50% Read

and varies the server count from 1 to 30. Network latency between servers is set to be 20. Half of the requests are read operations. The servers synchronize the write operation using two-phase commit protocol. Figure 6 shows the result.

It is clear that read operation and write operation reacts differently to the change. The reading latency keep decreasing with more servers added to the system. On the contrast, the writing latency decrease at the beginning, but start to increase again with more servers added. This also conform to our expectation, which is explained below.

Assume we have k servers, with the write operation to one server cause time x_0. When adding more servers, the average write latency will drop: $x_k = \frac{x_0}{k}$. On the other hand, assume the additional network communication latency caused by adding more server to be y. This value can be estimated proportional to k: $y = m(k-1)$. So

$$\Delta_{latency} = y + x_k - x_0 = (k-1)m - \frac{(k-1)x_0}{k}.$$

When k is small, we have $x_0 > mk$, so the average latency decrease. But when $x_0 <= mk$, the average latency will start increasing. Figure 7 supports this analysis. It can be seen with a higher network latency, the point of inflection comes earlier.

Finally we simulate the TACT consistency model. We simulate the communication between a cluster of 5 servers and 100 clients, using Eventual Consistency Model, Strong Consistency Model(2PC-Commit) and TACT with different parameters. We tried 4 set of TACT parameters. The first set [NumError = -1, OrderError = -1, Staleness = -1] simulates Eventual Consistency using TACT. The second set [NumError = 1, OrderError = 1, Staleness = 1] simulates Strong Consistency using TACT. The other two sets [NumError = 10, OrderError = 10,

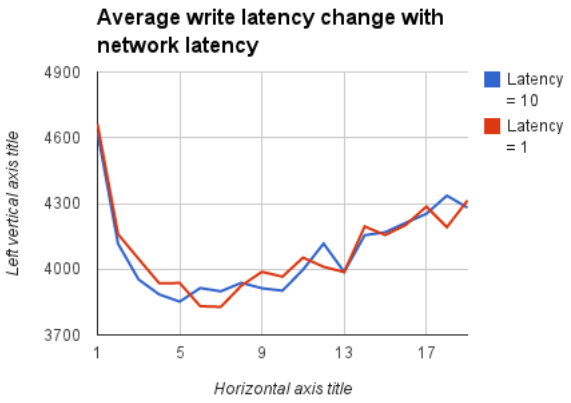

Figure 7: How write average latency change when changing network speed

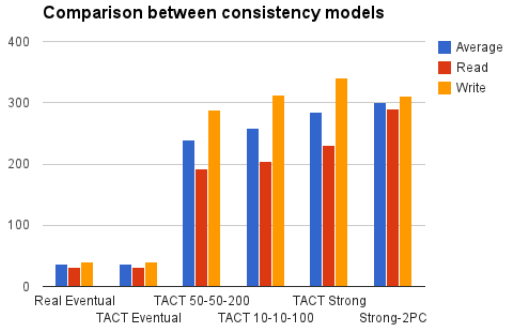

Figure 8: Comparison between Consistency Models

Staleness = 100] and [NumError = 50, OrderError = 50, Staleness = 20] is to compare with them. Figure 8 shows that when varies the parameter, TACT's average latency moves between eventual consistency and strong consistency. When setting a higher number of parameters, the latency drops. This result conforms to the result described in the original paper [4]: by increase the parameters, TACT model can decrease its synchronization frequency, so as to gain a lower latency.

3 Conclusion and future works

From these test result, it can be believed that the output result generated by the simulator actually reflect the trends that would happen in real world, which makes it available to conduct more complicated simulation work in the future. Also, by doing these test, we have established the performance basis. Any future simulation result will be compared to this.

The latest version of GDCSimulator can be downloaded at:

https://code.google.com/p/gdc-simulator/

Our future research of GDCSimulator will primarily focus on two directions:

- Performance and accuracy of simulation
- Design and simulate consistency models that fit GDC's need

One thing we primarily focus on is the performance. With proper simplification, GDCSimulator tries to simulate exactly the operations that happen in a real system. This gives the user more freedom to develop powerful simulation scenarios. However, in practice we also notice some performance problem caused by this. With thousands of nodes in a scenario, each loop takes considerable time. Decreasing the loop count, on the other hand, will degrade the accuracy. We plan to solve this problem by employing some statistical models to calculate the result instead of running each step and record the result. Also to maximize the utilization of multi-core machines, we also plan to add multi-thread support to the simulator, which will enable it to run simulation of multiple core in parallel.

Another work we will continue doing is the consistency model evaluation. With its special requirement, few existing consistency model fits the need of GDC system. Based on the characteristic, some new consistency models should be designed, tried and evaluated. GDCSimulator, which is designed for this purpose, could be expected to make this work much easier.

References

[1] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, June 2002.

[2] Google Inc. Google Data Centers - The Dalles, Oregon. `http://www.google.com/about/datacenters/inside/locations/the-dalles/`.

[3] Werner Vogels. Eventually consistent. *Commun. ACM*, 52(1):40–44, January 2009.

[4] Haifeng Yu and Amin Vahdat. Design and evaluation of a conit-based continuous consistency model for replicated services. *ACM Trans. Comput. Syst.*, 20(3):239–282, 2002.