# Redundancy with Request Canceling

Jianxiong Gao
University of Illinois
102 Sterling CT
Savoy, IL 61874
gao2@illinois.edu

Junle Qian
University of Illinois
606 Stoughton ST
Champaign, IL 61820
qian10@illinois.edu

## ABSTRACT

Large data centers today supports numerous interactive web services to users. For all these services, low latency has been proven to be one of the key requirements for high quality of service. As the services on these large data centers becomes more sophisticated, more collaborations among different nodes are needed, which factors up the influence of long latency outliers. The increasing popularity of virtualization in data centers exacerbates the problem by a factors of two to four[5]. With a higher demand on computational power and more network capacity in data centers, reducing the latency via redundancy can improve the overall performance. However, some redundant works can be avoided through request cancellation, thus the latency can be further reduced. In this paper we study the effectiveness of request cancellation to the overall performance of response time. We further investigate the impact of delay within data centers on the effectiveness of request cancellation.

## Keywords

Latency, Redundancy, Load

## 1. INTRODUCTION

As the world becomes more connected via Internet, large data centers are more popular than ever. For commercialized data centers, the most important concern is the quality of service(QoS). For interactive services, past work has shown that latency of the network has great impact on the QoS [2]. Large companies like Google, Facebook, Amazon etc. are constantly trying to reduce the overall latency for the service they provide. However with increasing amount of personalized information and the increasing demand of personalized service, the proportion of delay happens due to the queuing and computational delay happen inside the data center itself has become more important than the transmission delay. A simple web page request to data centers today may need the work of several hundreds of different nodes. It is predictable that in the future when more more complicated applications are deployed, the number of nodes traversed to accomplish a request will increase. The overall round trip latency is affected by the processing time of each node. The reply to the request may not arrive until the last node to finish. Thus the outliers for the delay from different nodes can badly hurt the overall performance. Virtualization makes things even worse. Even with the load balancing technique today, we cannot completely avoid these outliers. Past work has shown that from traces of a 3500 nodes Facebook cluster, the outliers are still common [1].

While there are increasing demand on consistent service latency, the network throughput within large data center networks has grown rapidly. Even though throughput cannot help to improve latency directly, this newly available resource gives us a chance to solve the outliers problem via redundancy. When issuing request inside the data center, instead of issuing to just one node, we issue multiple requests to different nodes. The first response from these nodes are taken and the rest are discarded. Apparently the total number of requests issued has increased. In the most basic scenario where we send two requests to two different nodes, we effectively double the load for the data center. Several past works has proven that under certain load, this method effectively reduces the overall completion time of the requests [4]. However the hardware demand for such systems are high. In fact the load of the system has to be lower than 50% in order for duplicate requests to work well. The overhead of such system may impose to much cost. Here in this paper we analyze the effect of request cancellation.

With request cancellation, whenever a request has been fulfilled, all other nodes will get an message such that they can cancel the same request, under the assumption that this request has not been processed yet. Clearly our approach requires more network bandwidth as we need to send out more messages. We also study the influence of round trip time latency on the effectiveness of request cancellation. It is clear that the higher the latency, the less effective cancellation can be. However there is no quantitative measurement in past works analyzing the impact of round trip time latency.

The latter part of this paper is organized as follow: Section 2 introduces our simulation structure. We present our results and analysis in section 3. Future will be discussed in section 4 and we conclude in section 5.

## 2. SYSTEM OVERVIEW

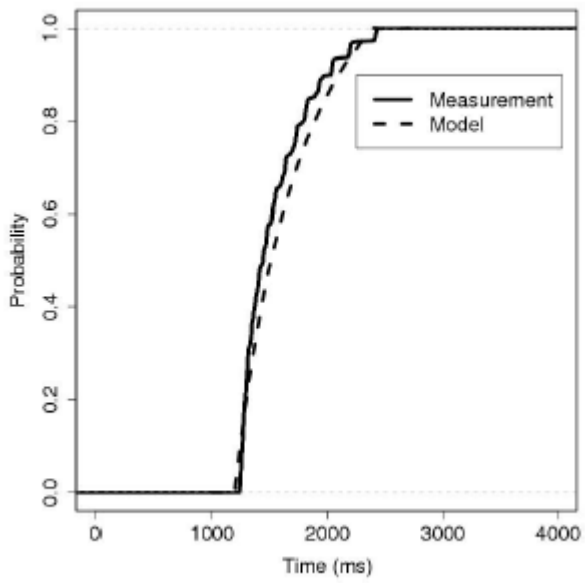


Figure 1: Disk Response Time Model

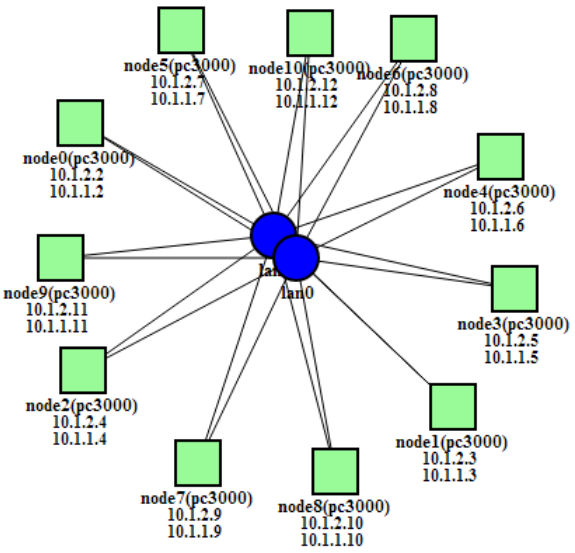


Figure 2: Emulab Topology

We first introduce our simulation of theoretical model. Our theoretical model is based on python code. The simulation consists of 10 worker nodes. The request assigned to each node is modeled as pure disk read and the response time is simulated using the model provided in [3], the cdf of measured and modeled disk response time is provided in figure 1. The modeled server generates requests using a Poisson process. The generated requests will be assigned to two randomly chosen worker nodes. The worker node queue up all the requests and processes them one by one. After a request has been finished, the worker node notifies the server. The server logs the completion time information. We further maintain the queue length information on the worker node. The assumptions we use here is that there is no network delay. Thus the result represents in ideal cases, the amount of improvement we can achieve by request cancellation. The results and analysis is provided in the next section.

We then build our model with Emulab. The topology is shown in figure 2. As shown in the figure, for each node we have two lan connections. Lan connection A is set to have minimal latency, which by real measurement varies from 0.1ms to 1ms. Lan connection B have varying delay set by our simulation input. Connection A is used for the server to assign requests to worker nodes. Connection B is used for the worker nodes to send cancellation messages. This may not reflect the real world scenario, however minimizing the server - worker delay helps to show the difference in request completion time with different Connection B delay, which can better illustrate the influence of network delay on the effectiveness of request cancellation. Intuitively when the delay of connection B approaches infinity, the model converges to ”redundancy without request cancellation”.

The server is always set to be node 0, and all the rest nodes are worker nodes. The server and client are both coded in C++. The server generates requests using a Poisson process, which are sent to two randomly chosen worker node.

The worker node has two threads. One listening for adding and cancellation of requests, maintains a queue structure of requests. All the added requests are queued at the end of the queue. The other thread takes the requests at the head of queue and processes it. The processing time is chosen uniformly from 500ms to 2000ms. After processing, it sends out a finish message to the server (node0) and a cancellation message to the other server which also gets assigned the same requests. The server records the processing time for each request, while the worker nodes keeps information about the maximum queue length. Those results are all written into the Emulab distributed file system, thus we can collect them at once. The simulation result and analysis is presented in the next section.

# 3. SIMULATION RESULT

## 3.1 Theoretical Model

We choose Poisson process to emulate the arriving of requests, because it is more realistic due to the variation of the time between two consecutive queuing events. The time between two consecutive requests is a random number generated from an exponential distribution. The arriving rate presented in this paper is the reciprocal of the mean value in this exponential distribution, and it effectively indicates the load on the queuing system. Theoretically we assumed infinite link capacity and zero link latency to verify the correctness of canceling unnecessary requests. Before the simulation starts, we first make request assignments for all workers. Every request will be assigned to two randomly chosen worker nodes, and this randomness is uniform.Each worker will keep a list, and each item in the list records the assigned request, its planned arriving time and ID of the other worker who is assigned with the same request. When the request is assigned, the planned arriving time of a request is generated according to Poisson process. Next, the worker nodes will start to process requests by going through a number

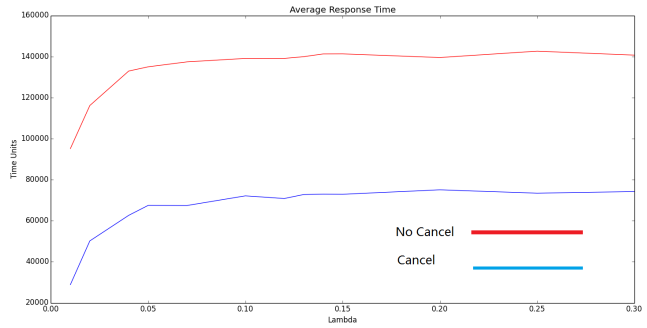


Figure 3: Maximum Queue Length for 10 worker nodes and 1000 requests

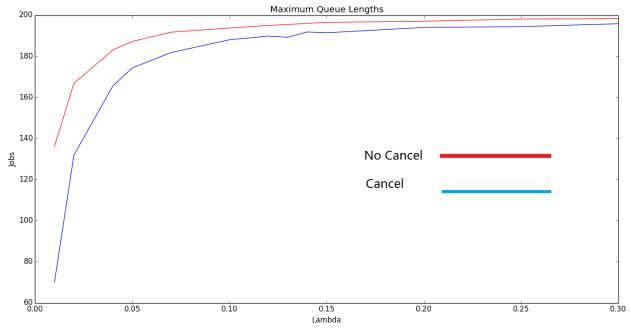


Figure 4: Mean Response Time for 10 worker nodes and 1000 requests

of rounds.We iterate rounds based on a virtual time, which is a local counter that will be incremented after each iteration.In each round, the worker will check its own list and check if a request has "virtually" arrived yet. If there is a request "virtually" arriving, the worker would remove this request from both the worker's own list and the list of the other worker. In the same time, this worker would check a set that records the response time of all finished requests. If the finished request already exists, it simply ignore this completion; otherwise, this worker would calculate the time elapsed between the current virtual time and the planned arriving time, and push this information to the set of time records. This simulation keeps running until all the worker lists are cleared. Note that this simulation pictures an absolutely ideal situation: no time spent on sending the canceling request. Although request assignments are randomly generated, they are virtually precomputed in the very beginning. We show the comparison of maximum queue lengths(Figure 3) and mean response(Figure 4) time below:

In Figure 3, we can observe that the mean response time is significantly reduced, which implies the effective improvement of overall performance in respect to latency. The ideal reduction percentage of mean response time is approximately 50%. Obvious difference between queue lengths can be found in Figure , while the system is not overloaded. In addition, we also discover that while the system is operating at high load, sending canceling requests no longer helps to reduce the latency by 50%, and the maximum queue length starts to approach the one without canceling request. Note that the canceling request that is sent out has nearly 0 harm to the system now, because we assumed the infinite link capacity(no congestion) and zero link latency(no overhead).

## 3.2 Emulab Model

Like mentioned in section , even though theoretical model is persuasive on the correctness of sending canceling requests, the model is not realistic.Two side effects must be considered. One is the negative effect on the pipe, such as link congestion some previous work speculated.The other important part to consider is the latency of sending a canceling request. Due to the nature of modern giant data centers, we first discuss about impact of additional latency for sending the canceling request. For realistic simulation, we build a all-to-all topology thanks to Emulab. Emulab is a network testbed which allows us to quickly build up a scalable network experiment.

### *3.2.1 Small Network Latency*

We first confirm if the realistic model confirms our theoretical model. The server and worker programs are programed distributedly, so that the server can no longer prepare for the request assignment before the system starts. The workers no longer share the virtual list with each other instantly. Instead, the worker and the server need to send messages to communicate with their targets in purpose fulfilling corresponding tasks. We define three different type of remote control messages that they will send to each other:

- *add* [requestId] [workerNode] [otherWorkerNode]
- *cancel* [requestId]
- *finish* [requestId]

The *add* message is sent from the server to inform a worker that a new request has arrived. requestId is the ID of the request and workerNode is the node who should get this notification. otherWorkerNode is the node ID of the other worker who also get the same request assignment. Thus, two workers will receive a request message wrapping the node ID of each other. After a worker receives the request, it will enqueue the ID of the request and ID of the other worker. Upon completion, this worker would send a *cancel* message to the other worker with an explicit request ID of the request to cancel. Then the worker would send a *finish* message to the server answering that this request has been completed. The server keeps a list of requests that has been finished. Upon receiving a *finish* message, the server will search for the newly arrived request in the finished request list. If this request has been finished by other worker, the newly arrived finish message will be discarded. Otherwise, the finish message will be translated into response time and recorded in the list. Upon receiving a *cancel* message, the worker searches removes the respective request from its local queue if it is found there. Otherwise, the worker discard this *cancel* message.

### *3.2.2 Impact of Network Latency*

From Figure 5 and Figure 6 we find that the performance improvement supported by the theoretical model is reproducible in a network environment with low latency related to the disk processing time. In Figure 5, we can see that

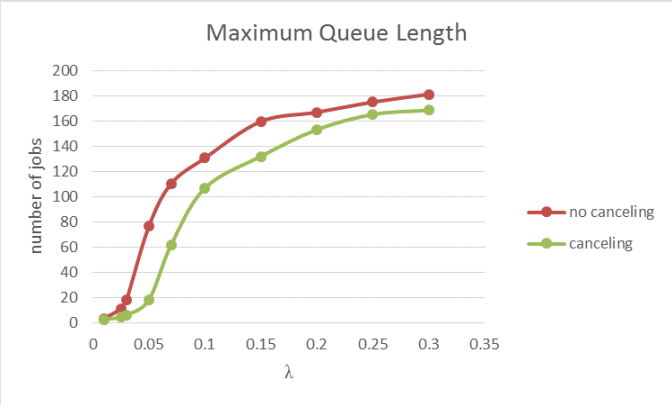


Figure 5: Maximum Queue Length with latency < 1ms, 1000 requests, 1 server node, 10 worker nodes

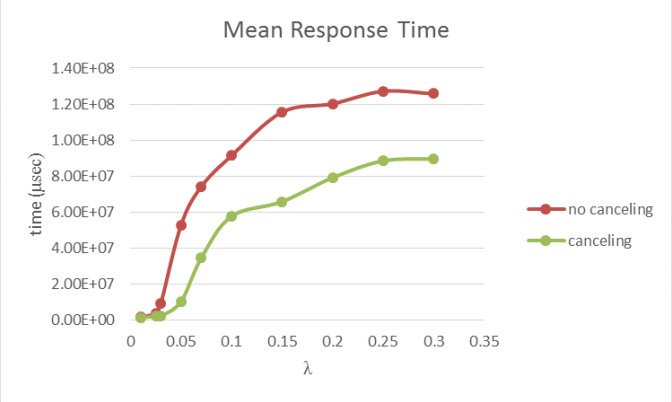


Figure 6: Maximum Response Time with latency < 1ms, 1000 requests, 1 server node, 10 worker nodes

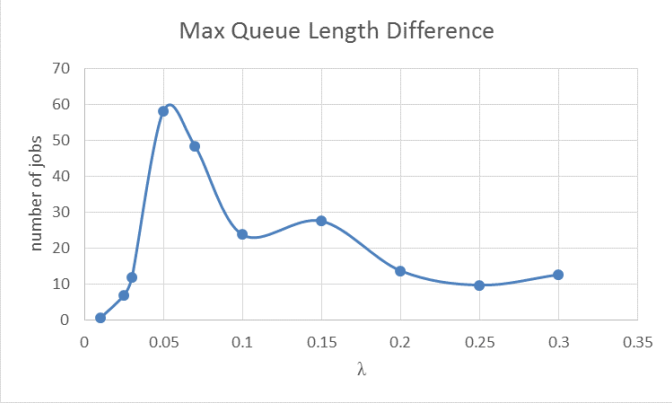


Figure 7: Maximum Queue Length Difference with latency < 1ms, 1000 requests, 1 server node, 10 worker nodes

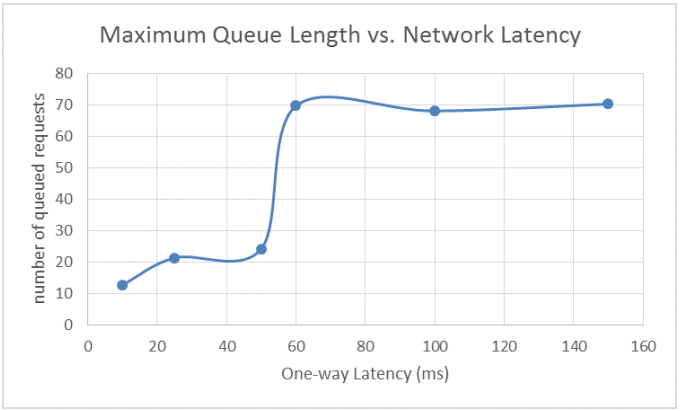


Figure 8: Maximum Queue Length ($\lambda = 0.05$)

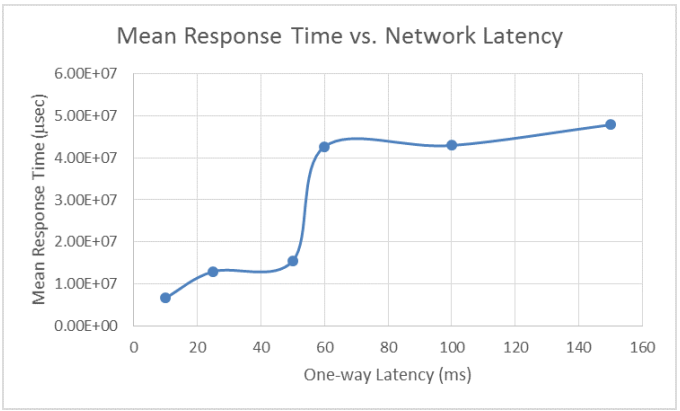


Figure 9: Maximum Response Time ($\lambda = 0.05$)

the gap between maximum queue lengths is closing after an arriving rate of approximately 0.2. In Figure 6 we discover that while arriving rate is floating between 0.1 and 0.2, approximately half of the response time is saved on average. We are interested how the system scale with network latency. We first selected an arriving rate that seems to get the most benefit of canceling request. In Figure 7, we analyzed the maximum queue length and decide to observe the mean response time at this fixed point.From the result, we select 0.05 arriving rate to be the fixed point.

We then vary the delay of Connection B from 10 ms one way delay to 150 ms one way delay. The simulation result of maximum queue length difference and mean response time are shown in figure 8 and figure 9 respectfully. It can be seen that under our set up condition, the maximum queue length and mean response time have a obvious increase after we adds an one way delay of around 50 ms. The mean response time with an added delay of 150 ms is 91% of the result of no cancellation. Thus we consider the effectiveness of cancellation can drop dramatically after some network threshold. However the relationship between the imposed load $\lambda$, network latency and request processing time is still not clear. This is left as our future work.

# 4. FUTURE WORK

We list here a few interesting points that can be further investigated. First of all, our model now uses an uniform distribution of processing time. However we can further improve our result with real world data. The model we have is just the simplest one at hand.

Secondly, in our model we do not consider prioritization of requests. With priortization, we can selectively insert requests to the location nearer to the head of the queue. Even without prioritization, we may selectively send out two requests with different labels, with one having lower priority which gets inserted nearer to the tail. The evaluation of such mechanism is interesting.

Lastly, in our model, we only consider one duplicated request. The impact of multiple duplicated requests is out of the scope of this paper. However it is also interesting to see the relationship between the number of duplicated requests and the resulted improvement.

# 5. CONCLUSION

The bandwidth within data centers today are growing at very fast speed. On the other hand, the computation speed is limited by the power consumption of CPU, the response time from memory and hard drive etc. Virtualization makes the variation of response time of a single node even larger in data centers. With more link capacity, we may use the technique proposed in this paper to eliminate the outliers in response time with a decreasing cost - a few more bytes for each request. Though the request cancellation method impose a strict restriction on network capacity and latency, with the new technology on networking recently, it will be a less important problem. The results shows that when we have enough capacity and link latency, the redundancy can indeed reduce the overall response time, thus provide better quality of service.

# 6. REFERENCES


[1] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica. Why let resources idle? aggressive cloning of jobs with dolly. *Memory*, 40(60):80, 2012.

[2] J. Brutlag. Speed matters for google web search. *Google. June*, 2009.

[3] A. S. Lebrecht, N. J. Dingle, and W. J. Knottenbelt. A response time distribution model for zoned raid. In *Analytical and Stochastic Modeling Techniques and Applications*, pages 144–157. Springer, 2008.

[4] A. Vulimiri, P. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker. Low latency via redundancy. *arXiv preprint arXiv:1306.3707*, 2013.

[5] Y. Xu, Z. Musgrave, B. Noble, and M. Bailey. Bobtail: avoiding long tails in the cloud. In *Proc. NSDI*, 2013.