15-440/15-640: Distributed Systems
Homework 3

REFERENCE SOLUTION
Last update: April 11, 2016

Question 1

A. What is the difference between "scale up" and "scale out"?

Solution: Scale up means making a single node faster by adding resources (for instance, processor cores, memory) to it; scale out means adding more nodes to the system.

B. Give two reasons why scale up may not benefit a particular application.

Solution:

- Hardware limit so that scaling up may be extremely hard (e.g. RAM slots)
- Too expensive.
- More difficult to share unused resources if load drops.
- Other reasonable answers.

C. Give two reasons why scale out may not benefit a particular application.

Solution:

- Higher adminitration cost (see question 2).
- Higher probability of failure and latency.
- May need to rewrite the application in some cases.
- Other reasonable answers.

Question 2

As a senior engineer at EBooks_R_US, you are asked to improve the scalability of the e-book service so that it can handle at least R read requests per second (R is specified below). Of course, you want this improvement to be achieved at the least cost. Currently, there are 15 Wimpy server machines that can each handle 10 reads per second. This gives an R value of 150 read requests per second. You receive the following quotes for new machines.

Machine Type	Target rate of requests handled per machine (reads/sec)	Price for the new machines (USD)
Beefy-A	20	400
Beefy-B	30	750
Wimpy	10	100

You can scale up by replacing existing machines. Or, you can scale out by adding more machines. An additional expense to be considered is that of system administration. The first 30 machines (of any capacity) can be administered by you personally; no additional personnel are required. After that, every 30 machines or part thereof (of any capacity) require an operator who costs \$10800.

For simplicity, you can ignore all other costs such as power, cooling, software licenses, etc. For the following values of R, determine the optimal (i.e., least cost) configuration change (i.e., combination of scale up and scale out). Explain your reasoning in all cases.

Solution: Three important points to address before we approach the questions:

1. In terms of the unit cost (defined as "USD/read request improvement"), Wimpy is the cheapest (10 USD), followed by Beefy-A (20 USD) and at last Beefy-B (25 USD). This reflects that scaling up can be increasingly expensive, as our demand of the target rate grows.

2. Scale out first before we scale up the existing machines. In a certain sense, we can consider the existing machines as free resources since we do not need to pay for them. Therefore, scaling them to Beefy-B, for instance, can cost us $\frac{750}{(30-10)} = 37.5$ USD per improvement of read request. Too expensive.

3. Hiring additional engineer (operator) should be avoided whenever possible. Having an additional operator (who costs \$10800) for 30 machines is equivalent to imposing an additional cost of \$360 on each new machine. For example, if this is imposed on the Beefy-A, it would make its cost per increase in request handling rise to $\frac{400+360}{2} = 380$ dollars per 10 reads (i.e. \$38 per read request). This is

higher than all three machine types, so we should avoid this when we can. (This reflects the fact that, although scaling-out is cheaper, in real life we do need to take care of other non-hardware costs.)

A. For $R = 300$, what is the optimal configuration? What is its cost?

Solution: Simply scaling out by purchasing another 15 Wimpy machines, making 30 in total. The cost is then $15 \cdot 100 = 1500$ dollars. This is the cheapest because we chose the cheapest possible without hiring new operator or changing any existing machine.

B. For $R = 900$, what is the optimal configuration? What is its cost?

Solution: Note that $900 = 30 \cdot 30$, which means we are going to need an additional operator unless we have exactly 30 Beefy-B machines. By observation 3, we should avoid new hire whenever possible, so the optimal configuration is to scale up the current 15 machines to Beefy-B, and scale out additional 15 Beefy-B machines. Total cost is $30 \cdot 750 = 22500$ dollars

C. For $R = 1000$, what is the optimal configuration? What is its cost?

Solution: We now need at least one additional operator anyway, so our main goal is to find the optimal configuration but without the need to hire a third engineer to administer the cluster of machines. In particular, we should keep the existing 15 Wimpy machines unchanged (they do not incur additional cost) and fill the rest of 850 requests handling requirement using the 45 machine spots left to contain some combination of Beefy-A, Beefy-B and Wimpy.

The reasoning is very simple. We should first fill the whole 60 available spots with Wimpy and then eliminate the gap by scaling up a portion of the machines from Wimpy to the next cheapest choice— Beefy-A. The optimal configuration should be 20 Wimpy machines (15 of which are existing ones) and 40 Beefy-A machines, giving us a total cost of $5 \cdot 100 + 40 \cdot 400 + 10800 = 27300$ dollars.

Question 3

Suppose you have access to a compute cluster that can run MapReduce jobs, as well as a supercomputer that can efficiently run MPI jobs. You would like to pick the right tool for the right job. For each of the following, state whether the use case is a better fit for MapReduce, MPI, or neither. Explain your answer in each case. If MapReduce is a better fit, briefly describe what mappers and reducers do. If MPI is a better fit, briefly describe why it wins over MapReduce.

A. Compute the average of all pixels in a 1000×1000 image

Solution: Neither. The dataset is too small for both, so using MPI/MapReduce will lead to a hight cost. (Other valid justifications are also accepted)

B. For each of 10^6 images, compute the average value of all pixels in an image. Each image is of size 1000×1000 pixels.

Solution: MapReduce. The mapper will process each image and reducer will store the result to GFS.

C. Process an image of size 10^6 by 10^6 pixels, where the processing of each pixel depends on all of the neighboring pixels, and the processing involves several thousand iterations.

Solution: MPI. Processing pixels depends on neighbor pixels. Fine-grained.

Question 4

A cloud storage service allows creation and deletion of blobs (blob = "binary large object"). Once created, a blob cannot be modified. The storage service uses one master node and multiple data nodes. Creation of a new blob occurs at the master node. A unique identifier is first assigned to the blob. The blob is then broken into *chunks* whose size is normally distributed, with a mean of 1 MB and a standard deviation of 100 KB. Chunks are content-addressed. In other words, the SHA-256 hash of a chunks content is used as that chunks unique identifier. How to reconstruct the blob from its chunks is documented in the *recipe* of that blob, which is stored at the master node.

A. State one advantage and one disadvantage of using content addressing for chunks. *(Hint: think about deletion of blobs.)*

Solution:

- Advantage:

  1. Deduplication. There will never be more than one copy of an identical chunk in the storage system. (Two identical chunks have the same content address). So it can save the storage space and store data very efficiently.
  2. Search is fast and provides an assurance that the retrieved document is identical to the one stored, so we can verify things we get, even from an untrusted server.
  3. ...

- Disadvantage:

  1. Since there is only one copy of identical chunks in the cloud storage, when some blob containing that chunk needs to be deleted, we have no idea whether we should delete that chunk or not since that chunk might be shared by other blobs. (might require traversing the blob list in the master node)
  2. ...

B. Consider two alternative designs. The first design maps chunks to nodes based on the low order bits of the chunk identifier. For example, if there are 4096 data nodes, the low order 12 bits of the chunk identifier determine the mapping. The second design treats the data nodes as a DHT that is addressed by chunk identifier. Which of these two designs is better? Justify your answer.

Solution: Either choice is acceptable as long as the student provides reasonable justifications.

- First design is better:

  1. The lookup operation of first design is faster than DHTs. (e.g. Chord with finger table requires contacting $O(\log N)$ nodes in a N-node system, first design only requires contacting $O(1)$ node)
  2. ...

- Second design (DHT) is better:
    1. DHT is better for a new node to join or leave (e.g. using consistent hashing) since only a small fraction of chunks need to be redistributed. However, first design requires most chunks to remap when a new node joins.
    2. DHT makes the service more scalable.
    3. ...

Question 5

A client/server model and a DHT are two alternative ways of organizing a large collection of read-only data from many users, such as encrypted backup snapshots of their home PCs. In this context, answer the following questions.

A. State and explain why you might prefer the client/server model.

Solution: There are several potential reasons:

- Simpler model is easier to implement and manage.
- Clear responsibility between server and clients.
- No need to traverse the nodes (which happens in the DHT) design.
- (Potentially) better security protocol between server and clients.

B. State and explain why you might prefer a DHT.

Solution:

- Robust to failures.
- Scalability.
- More balanced load (no one is “too hot”).
- Low starter cost.

C. In a real world situation, how will you decide between these alternatives?

Solution: *Either way is acceptable, as long as the answer makes valid assumptions about what aspect is valued more/given higher importance in the given scenario*

For example, for a startup/company offering a particular service, server-client model is probably better; for information shared among a group of people (with no real "manager"), DHT can be a good choice.

Question 6

Which of the following workloads would MapReduce be good for? In each case, provide a couple of sentences justifying your answer.

Note: This question asks whether or not MapReduce is a good choice. Even though it might be possible to use MapReduce for some cases, it might not be a good choice comparing to alternative distributed computing approaches like MPI.

A. Building a real-time news aggregation service that provides up-to-the-minute news updates. It should be designed to parse feeds from multiple news websites and refresh its index in real-time.

Solution: MapReduce is not good for this workload.
This workload requires real-timeness. MapReduce is usually not fast enough for that.

B. Building an interactive data exploration tool which lets users provide their own queries and rapidly see partial results from their queries without waiting for the query to run over an entire dataset.

Solution: MapReduce is not good for this workload.
MapReduce is not good for interactive applications and it does not support partial results.

C. Building a high-frequency trading application. You've been asked to architect a high-frequency stock trading application which needs to consume stock ticker updates and provide quick decisions for trades.

Solution: MapReduce is not good for this workload.
This workload requires real-timeness. MapReduce is usually not fast enough for that.

D. Building a database of features from images for face recognition. The input is hundreds of millions of user uploaded and tagged photos, each stored as a file in the local file system. For each image an algorithm must run that extracts features for training face recognition algorithms in a later stage.

Solution: MapReduce is good for this workload.
Each mapper could just look at a subset of the photos, without worrying about other photos. This application is also not time-critical and not iterative. MapReduce is great for this kind of applications.

E. Crawling through billions of log entries in log files, one per line of input, across thousands of websites and counting the number of unique visitors (unique by IP address) that access those websites. This will feed daily, weekly, and monthly reporting systems that provide website analytics.

Solution: MapReduce is good for this workload.
Each mapper could just look at a subset of logs, without worrying about other logs. And reducers can simply add the counts up. This application is also not time-critical and not iterative. MapReduce is great for this kind of applications.