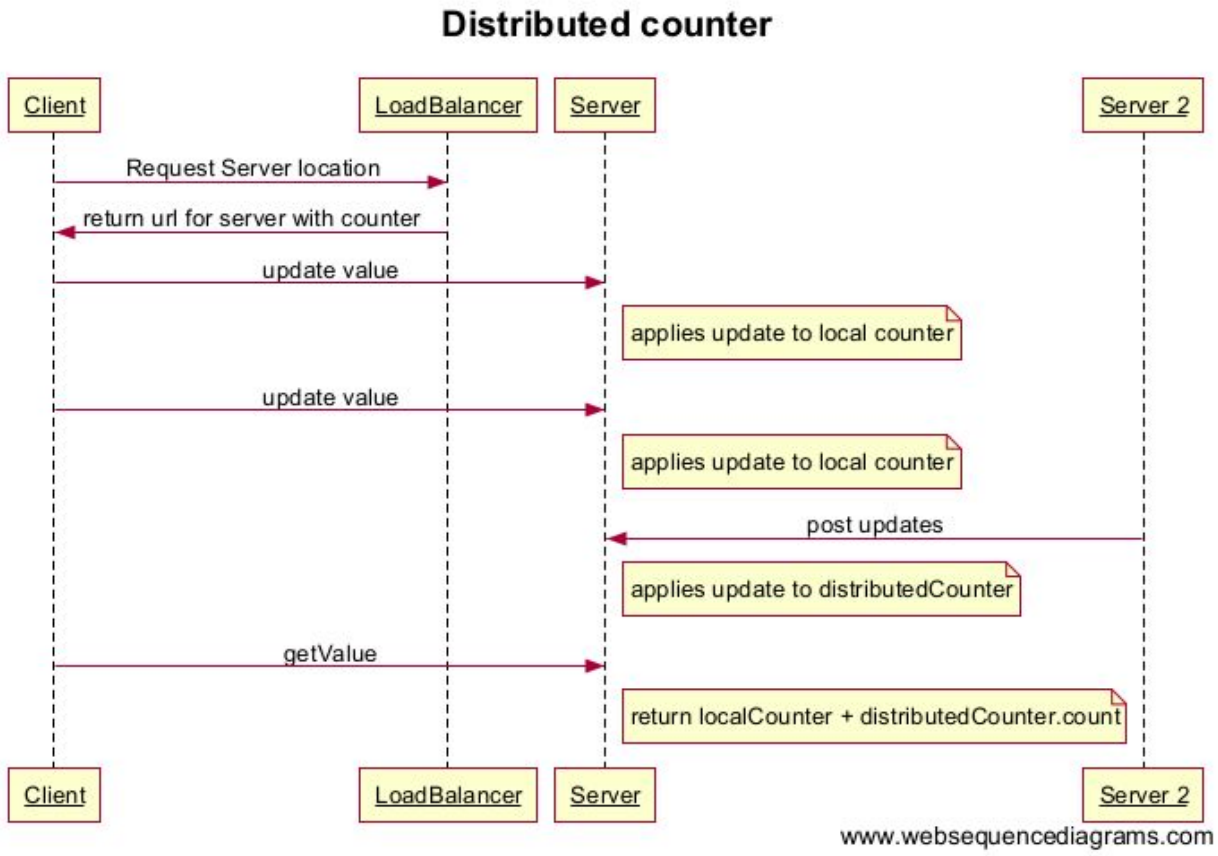
Distributed Counter Code Test

Summary

I built my distributed counter using multiple counting servers and a load balancing server.

When a client wants to modify the counter they contact the load balancing server.
The load balancing server provides them with the URL of a counting server. Using this URL they can modify the counter. they can also read the counter from this server.



Distributed Architecture

Keeping in line with the spec the entire architecture runs on a REST framework. I used jersey. the reason I choose to use is because it seems very standardized and to have very good support for unit testing.

Counting Server(maven artifact server-impl)

Every counting server is comprised of 2 internal counters.
The first is its local counter and the second is it distributed counter.

local counter

Local Counter is responsible for keeping track of the direct updates it has received from clients. This is implemented as an AtomicInteger. It ensures there there is no data corruption, and there is very little contention on this object.

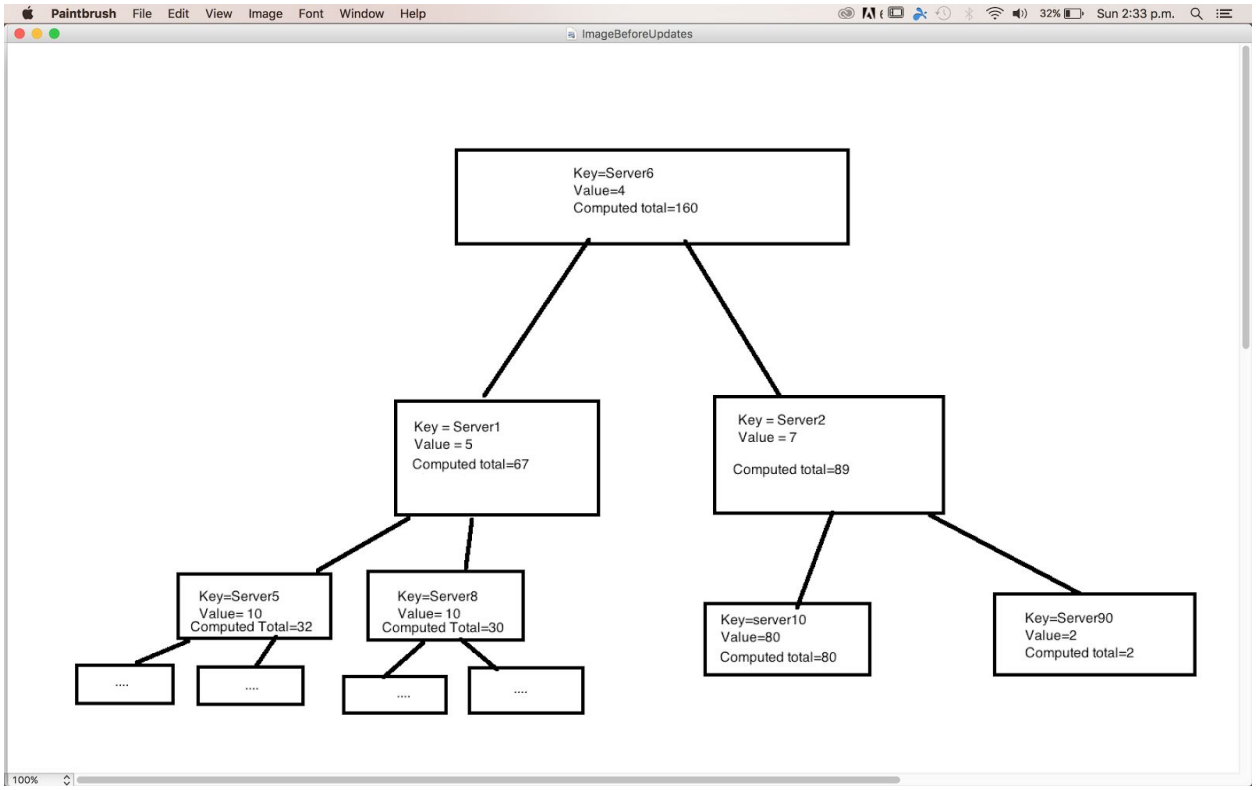
Potential performance improvements:

Could use a low contention data structure like below although the idea in my design is that there are only a limited number of clients connecting to a single server.

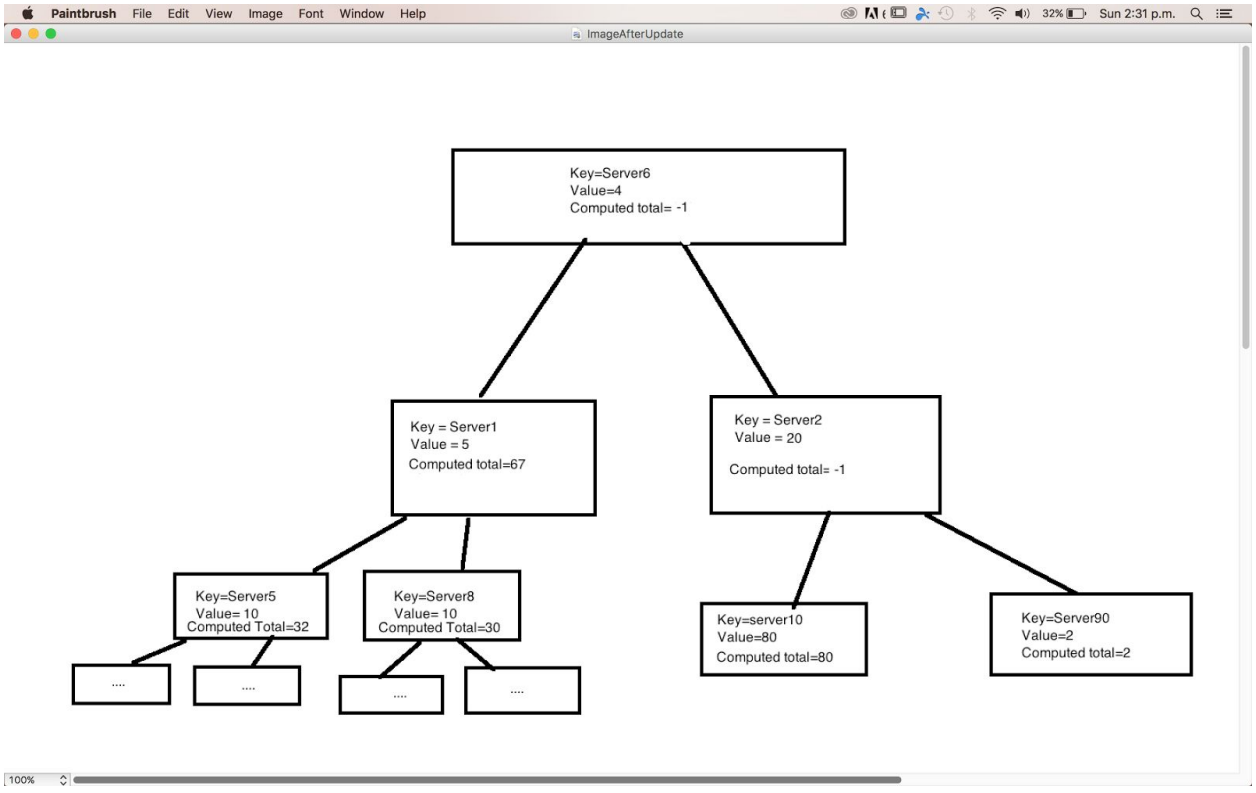
Distributed Counter (CountCollatingService.java)

Distributed Counter is implemented using a sorted tree structure to ensure no contention or corruption of data. Each node in the tree contains a key(Server identifier), value (current count of server), computed value (the sum of all this nodes children and its own value).
When servers are posting updates to this distributed counter data structure there will be no contention as each update will only impact its own node.
When a getValue request is made the tree begins computing the computed values in the tree. Next time a request is made to the tree for the count it will be able to return it instantly.
If an update is received then it will change the value of that node and invalidate the computed value of all its parents. However it will not invalidate branches of the tree not related to this node.
In the images below there is a tree, after getValue has been called on it. In the second image is a tree which has received an update. The tree ensures that minimal recomputation needs to be done next time the client requests the count.

Before update



After update



It goes without saying that this object performs when alot of writes happen in close succession and a lot of reads happen in close succession.

Sharing updates among servers.

Servers will post their updates to other counting servers every x seconds.
This value can be configured through the context.xml

Recovering after failover

When *Server A* comes online it contacts the load balancer server to get a list of all the other counting servers. It send a request to all of these servers, asking them if they have any knowledge of its count. If the server died in production it will be able to get its previous count back from one of these servers accurate to x seconds. X being the frequency at which it publishes to other servers.
As long as all the servers don't go down the there should not be a loss of data.

Assumptions:

1. The client does not need the exact counter value all the time.
2. All the servers will not go down at the same time.
3. A client stays connected to the same server for a long time.
4. Lots of writes normally happen at the same time.
5. Lots of reads normally happen at the same time.

Capabilities of the system:

1. Both updating and reading the value is a non-blocking operation
2. If the accuracy of the counter is not very important then the amount of intra server IO used can be set quite small.
3. If a server crashes It can get its counter value before death from another server and continue on as normal. Although this operation uses up a lot of IO.

How to Run

From the application directory

1) Run the script startupAllServers.sh
    a) Starts the load balancing server
    b) Starts 2 counting servers
2) run the client-impl*.jar from the command line
3) modify the values etc...