Real time trajectories with Hazelcast

Distributed Systems Seminar

Joosep Rõõmusaare

December 19, 2013

Introduction

Mobile network carriers want to analyze the cellular network - e.g. how many people are connected to the some cellular telephone site at some time? For this cellular telephone sites have collectors collecting cell data events about the users connected to them. One cellular telephone site can own one or more cells, what cover some area, where telephone could connect to that site.

Collector stores each day last 24 hours of data into tar file. Tar is given to the Trajectory database (TrajectoryDb), from where history service can make search request for the past cellular activity. But as data is written into tar after each 24 hours, then TrajectoryDb can't take request quite real time, because its data is not up to date from the last tar creation until next tar creation. For that there is need for storing that missing data somehow, so there would be possible to make request for them in real time.

One way to store the events, what are not in the tar yet, is to hold them in memory. So when the next tar is created, it can remove events what are then covered with tar files and always keep taking in new events from the collector. Goal of this report is to analyze if Hazelcast [1] could be used holding up to 24h of events data in memory.

Trajectory database

TrajectoryDb goal is give quick responses for trajectory queries. It creates it's own indexed data format from tar files, where it can find fast results for given requests. Tar files contain cell data events and event contains location and time data for some pseudonym. Pseudonym is pseudonymized cell user identificator. And trajectory is chronological list of all events for one pseudonym. And with the use of TrajectoryDb it is easier to make request for events after their pseudonyms, but only for the data what is older than up to 24 hours as was explained in the introduction.

Hazelcast

Hazelcast is in-memory data grid which provides a clustering and highly scalable data distribution platform. It should be very fast (thousands of operations per second), fail-safe (no data losing after crashes), dynamically scaled when adding new servers, and easy to use. It is a peer-to-peer solution, what means that there is no master node and with that no single point of failure.

Even if we only look at the Community version of Hazelcast, there is also Enterprise edition. Enterprise edition adds elas-

tic memory, what is Hazelcast with off-heap memory storage to avoid garbage collection, what will result in more predictable latency and throughput. There is also Hazelcast Security framework, what helps to authenticate both cluster members and clients and do access control checks on client operations. And with Hazelcast you can use their Hazelcast Management Center, what can be used for monitoring and managing Hazelcast clusters. For Community edition management center can still be used, but is limited to only two nodes.

Hazelcast can be used for different things. Sharing data or states among many servers, Distributed cache of data, providing secure communication among servers, Partition of in-memory data, Distributing workload onto many servers, take advantage of parallel processing, and providing fail-safe data management. And in this report it is used for most of these use cases. As it is used for clustering our real-time trajectories application, it also takes care of partition of our in-memory data. With that Hazelcast will distribute workload over each node and when there would be some request for our data,then it would run that query in each node of our cluster at the same time. Finally as Hazelcast backs up data of each node over other nodes, then it provides fail-safe data management.

Real-time Trajectories

For the last 24h of cellular telephone event data there is implementation of TrajectoryDb what will receive new cell data straight from collector in cell site. It will hold last 24 hours of data in memory until tar containing that time period is available. Current solution has all the data in one service. This is illustrated by Figure 1. And if this service crashes then there isn't possible to make any request for trajectories (this includes real-time events and events from tar) until TrajectoryDb is started again. Then it is again possible to make request for time period containing in tar files and new real-time data is received again from collector, but events from the last tar until restart is missing. There could be work-around where collector or some other subsystem could resent that data, but it would be inconvenient. So better system for real time data would be welcomed.

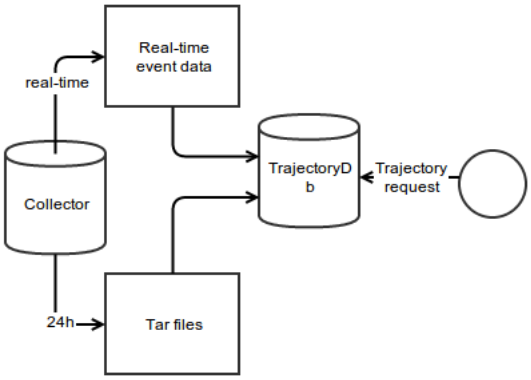

Figure 1: Current TrajectoryDb

Using Hazelcast

If between collector and real-time trajectories implementation of TrajectoryDb would be Hazelcast cluster, then most of the problems with real-time trajectories could be gone. Collector could implement Hazelcast client called EventProducer instead of sending real-time events straight to the TrajectoryDb. EventProducer will send real-time events from collector to the Hazelcast cluster, what has multiple nodes what will hold and distribute event data. And on real-time part of TrajectoryDb would implement EventConsumer, what can make requests for events after their pseudonyms

and also tell to the cluster what are the oldest events to hold in the cluster. This is illustrated by Figure 2.

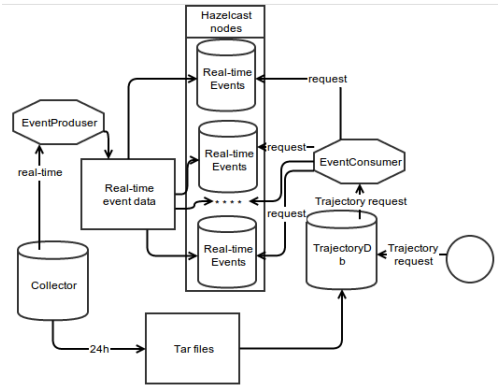

Figure 2: TrajectoryDb with Hazelcast

As Hazelcast can be used to distribute trajectory events over different services, it takes care of our main concern: that with crash of service, data would go missing. With the new system when TrajectoryDb itself crashes, then real-time services are still running and taking in cell data events from collector. And if TrajectoryDb is back on-line, it can start making again real-time queries with EventConsumer.

If some real-time data node from cluster crashes, then Hazelcast have backed that node data up to other nodes of that cluster. So other nodes that have crashed node data will start using that data and again that data would be distributed between other nodes. So as long as multiple nodes don't close at the same time, there isn't any fear that some data would be go missing.

Both EventConsumer and EventProducer clients could connect at the beginning to single node from the cluster, but after initial connection they will know about other nodes. And if that node would go down, then it will be automatically replaced with another node from cluster for main connection between client and service. When making requests from real-time events cluster or sending data to the cluster, then Hazelcast will take care of distribute request over all nodes, so there is not any nodes where the load is higher.

Testing Hazelcast

For Testing how could Hazelcast cluster work with real-time event data, there were created simple classes EventConsumer, EventProducer, EventClusterInstance and InstanceTest. The latter creates instance with multiple number of EventProducers, what will start sending events to the cluster. Code for the java project is available in the 2.

#	Members	Entries	Entry Memory	Backups	Backup Memory
1	192.168.12.107:5701	433840	103.42 MB	430241	102.56 MB
2	192.168.12.107:5702	434653	103.61 MB	438439	104.52 MB
3	TOTAL	868493	207.03 MB	868680	207.08 MB

Figure 3: Events per cluster nodes

Table from Figure 3 is taken from the Hazelcast Management Center. That result was created when two InstanceTest and one EventConsumer classes were running. EventProducers created 800000 events in less than four minutes. In six minutes there were over one million events but for then java heapspace run out and tests crashed. When InstanceTest classes were draining free memory, then EventConsumer did after every pause of one second requests for the count of events in the cluster and after that the oldest event. For that it has to iterate through all the events. When event count was circa 800000, it did oldest event request in 5-7 seconds. Times when it took more time, 15 seconds or more, was when one node got killed and added again to the cluster and Hazelcast tried to balance nodes.

The requests should be faster for more specific requests, where results are not needed to iterate through and the search would use Hazelcast search methods, where indexing is available.

When making request for searching pseudonyms from event data with the size on 700000 average request time was 4-5 seconds. pseudonym was randomly generated from 0 to 10000. and request were also randomly taken from that range. When cluster were indexed after pseudonym then request times were a lot faster. For 800000 events request took less than 25 milliseconds and average events to return were in range of 50-80 events for searched pseudonym.

Figure 3 shows how both nodes are balanced with event count. But backups count is not the same as entries count. That may be happening because producers are spamming nodes and number were already changed when management center made the request or it just don't have time to do backups correctly on the fly.

Conclusions

Test seems to show that it can take in a lot of data in short times. And even if harder requests took more time, then more specific requests would be faster. Test weren't good to show just how good would Hazelcast be. There could have been more different test to do, but it still showed that it's possible to use it for our real-time data. It has at least distribution (and with that fail-safe) advantage over current real-time TrajectoryDb solution. And also there would not go any data missing when for some reason TrajectoryDb have to have a restart.

Also test showed when cluster is configured correctly, then requests for information are fast. Which means that Hazelcast should not be puddle-neck for trajectory requests.

References

[1] Hazelcast. Hazelcast homepage. 12 2013. URL `www.hazelcast.com`.

[2] Joosep Roomusaare. Realtime trajectorydb dummy test. 12 2013. URL `https://github.com/joosep/hazelcast-demo`.