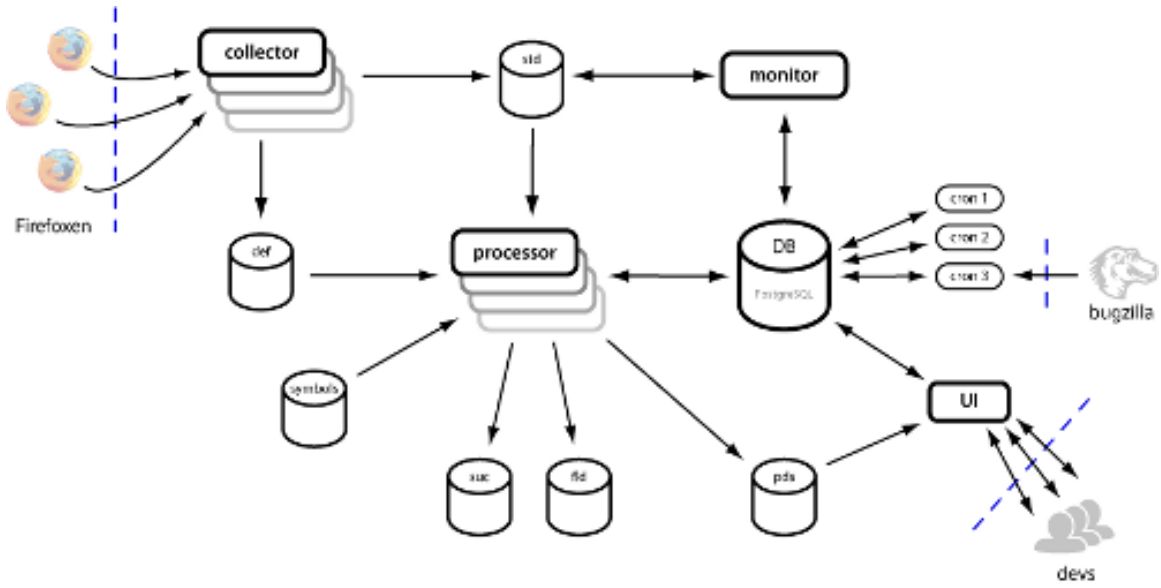
Socorro, is a server for collecting, processing, and displaying crash reports from clients using the Breakpad libraries. The current focus of Socorro development is to make a server which can accept crash reports from Firefox [1].



Currently, Socorro has pythonic collectors, processors, and middleware that communicate with HBase via the Thrift protocol. There is a class named hbaseClient.py that contains several pieces of the business logic such as creation of HBase rowkeys based on ooid, and the methods for putting or retrieving data in HBase and manipulating queues and metrics counters.

One of the biggest limitations of the current architecture is that it is very sensitive to latency or outages on the HBase side. If the collectors cannot store an item in HBase then they will store it on local disk and it will not be accessible to the processors or middleware layer until it has been picked up by a cronjob and successfully stored in HBase. The same goes for processors, if they cannot retrieve the raw data and store the processed data, then it will not be available[2].

In current implementation a collector stores the crash report into HBase and puts the id of the report to Hazelcast Distributed Queue. The processors polls the Queue and start to process the report. Given the problem stated as HBase outage, Daniel Einspanjer from Mozilla contacted us for the possible usage of Hazelcast Distributed Map, in front of the HBase. The idea is to replace the thrift layer with Hazelcast client and store the crash reports into Hazelcast. An HBase Map store [3] will be implemented and Hazelcast will write behind the entries to HBase via that persister. The system will cash the hot

data and will continue to operate even if HBase is down. Beside HBase, Socorro uses Elastic Search to index the reports to search later on. In this design Elastic Search will also be fed by Hazelcast map store implementation. This enables the Socorro to switch the persistence layer very easily. For small scale Socorro deployments, HBase usage becomes complex and they would like to use Elastic Search as the only storage.

Current Socorro application stores about 2.6 million crash reports per day. During peak traffic, it receives about 2.5K crashes per minute.

So we decided to make a POC of this design and run it on EC2 servers. The details of the POC requirement can be found at [2]. We have implemented the POC application. It is available under Apache 2.0 license at https://github.com/fuadm/socorro-hazelcast-poc/. Even though we have implemented the Elastic Search persistence, we didn't enable it during the first demo. The second one will include the persistence and will address the scalability issues.

To run the application you only need to have the latest Hazelcast and Elastic Search jars. The entry point to the application is com.hazelcast.socorro.Node class. When you run the main method, It will generate one Collector and a Processor. So each node will be both a Collector and a Processor. The Collectors will simulate the generation of crash reports and will submit them into Data Grid. The Processors will receive the newly submitted crash reports, process them and store back to distributed Map. If you enable the persistence, Hazelcast will persist the data to the underlying storage.

We run the application on 50 Node EC2 servers. In our simulation the average crash report size was 4MB. We had to store as much crash reports as we can, so we decided to go with largest possible memory which is 68.4 GB that is available at **High-Memory Quadruple Extra Large Instance.**

Hazelcast Nodes uses Multicast or TCP/IP to discover each other. On EC2 environment multicast is not enabled, nodes should be able to see each other via TCP/IP. This requires to pass the IP addresses to all of the nodes somehow. At the article Running Hazelcast on a 100 node Amazon EC2 Cluster we described a way of deploying an application on large scale on EC2. In this demo we used a slightly different approach. We developed a tool, called Cloud Tool that starts N number of nodes. The tool takes as an input the script where we describe how to run the application and the configuration file for hazelcast. It builds the network configuration part with the IP addresses of nodes and copies (scp) both configuration and script to the nodes. The nodes, on start, waits for the configuration and script file. After they are copied all nodes execute the script. It is the scripts responsibility to download and run the application. This way within minutes we can deploy any hazelcast application on any number of nodes.

With the similar way we deployed the Socorro POC app on 50 node m2.4xlarge instances. By default each node generates 1 crash report per second. Which makes 3K crash reports per minute in the cluster. In the app we are listening to Hazelcast topic called "command". Whenever we publish a message like s2, all nodes start to generate 2 crash reports per second. This way we can increase and decrease the load at the application while it is running.

We did the demo and recorded all the steps. We used the Hazelcast Management Center to visualize the cluster. Through the tool we can observe the throughput of the nodes, we can publish messages to the topic and observe the behavior of the cluster.

[1] http://code.google.com/p/socorro/
[2] https://wiki.mozilla.org/Socorro:ClientAPI
[3] http://hazelcast.com/documentation.jsp#MapPersistence