# Cricinfo System Design

## Discussions

### Know about the traffic

To know about a particular site deciding how many requests approximately it’s handling approximately and where the requests are coming. We can use below website to get those details.

* Similar web
* Alexa

This helps us distributing our app and placing data centers on region and locations. Choose our server location in AWS.

Also it would be deciding factor to get to know from where these requests are coming. From Mobile, Desktop, Gaming console etc. That way we can model our system.

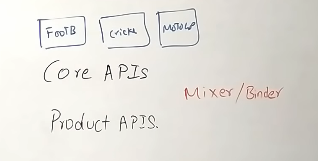
### Two API

Being Cricinfo a system which serves for many different games and each game having its own independent characteristics, just a simple MVC system exposing some end point can’t serve neither can scale well.

So internally the system can be granularly divides into 2 API parts

* Core API – APIs servers for different games/services
* Product API

We will have a Mixer/Binder which will bind the response from core APIs and return to Product API which may render for Home page etc.



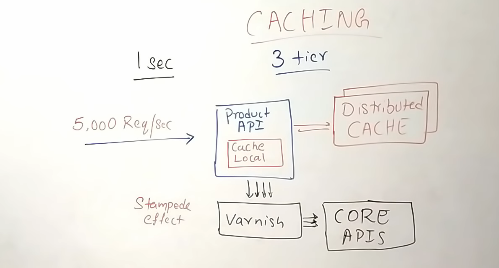
Binder also parse the query send to Product API (JSON request) and understands which service(s) it needs to call (Binder has the mapping of which microservice has to be called for which request) with what parameter to serve the query

So if a Product request comes which needs to call 5 different APIs, Mixer understands it and sends the job to 5 different services which can run in **parallel.**

### 3 layers for caching

When we have lots of data to serve, caching is the way.

Caching reduces the load on the servers. Cricinfo has 3 layers of caching.



Cricinfo uses **EhCache** for local cache and **Memcache** for distributed cache. Very hot data (limited) can be stored in Local cache i.e. Product API server cache.

**Varnish** has a feature called request collapsing (Stamped effect). If the same request is used a lot (say 2000 req/s) for the same API.

Say we got a request for score and get it from core API and updated the local cache say for 1 sec. At the end of 1sec the cache will be invalidated and request will go to Core API to fetch the data. By that time many other request will comes to get the same data (ex. cricket score).

If for the same data we hit the core API many times, it’s bad. So we use **varnish.** For a span of sec or millisecond if same request is coming for same data (ex. score), varnish will **collapse all of the request to one request** and only one request will hit the core API and response will be send to as many request came.

This situation observes when we have heavy request rate (ex. 5000 req/sec).

### Debug

With all these different caching layer placed, it will be hard to really get what happening with the request and from where it served. So on response we can add some extra metadata information defining which particular cache gets the data or the requests were collapsed and get response from core API.

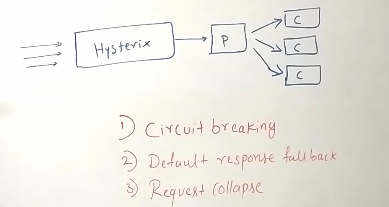
That way it helps to monitor or debug the bug in API.

### Hysterix

Many things might go wrong when a product API called to core API.

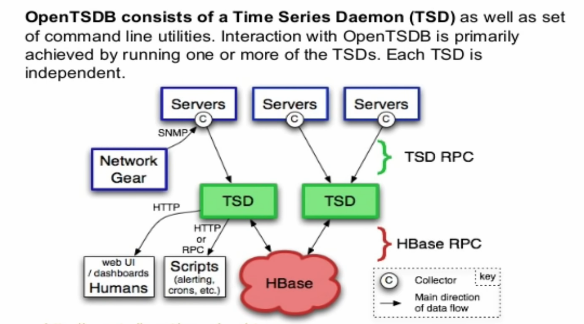
Hysterix serves below of the purpose.

* Circuit breaking
* Default Response Fallback
* Request Collapse – Hysterix can also makes request collapse on Product API end.



### Monitoring all these API

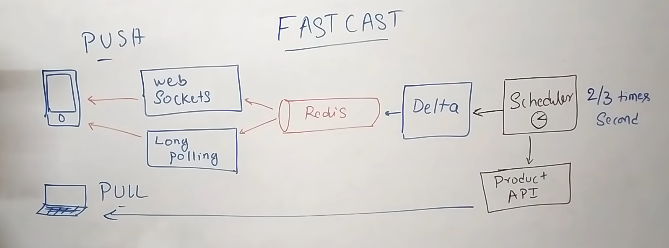
Cricinfo uses **OpenTSDB** to monitor all these APIs logs, to monitor their API performance at a given point of time. OpenTSDB builds on top of Hadoop and HBase so it’s high performance and also scalable.



OpenTSTB supports Griphana, Elastic search.

### Sending the data to client

We need to use Web Socket (preferably) or Long polling for pushing the data to client.



Here is a scheduler wakes up 2/3 times a second and call to the APIs gets the data and push the data. Delta is a service will hold the previous state, takes the current state and find the different and that difference will be pushed to Redis.

Every game happening and a given time will have their own queue.

The device which doesn’t support Web Socket may fallback to long polling or polling.