Improving HBase availability in a multi-tenant environment

James Moore (jcmoore@hubspot.com)
Kahlil Oppenheimer (kahlil@hubspot.com)

Outline

- Who are we?
- What problems are we solving?
- Reducing the cost of Region Server failure
- Improving the stochastic load balancer
- Eliminating workload driven failure
- Reducing the impact of hardware failure

Who are we?

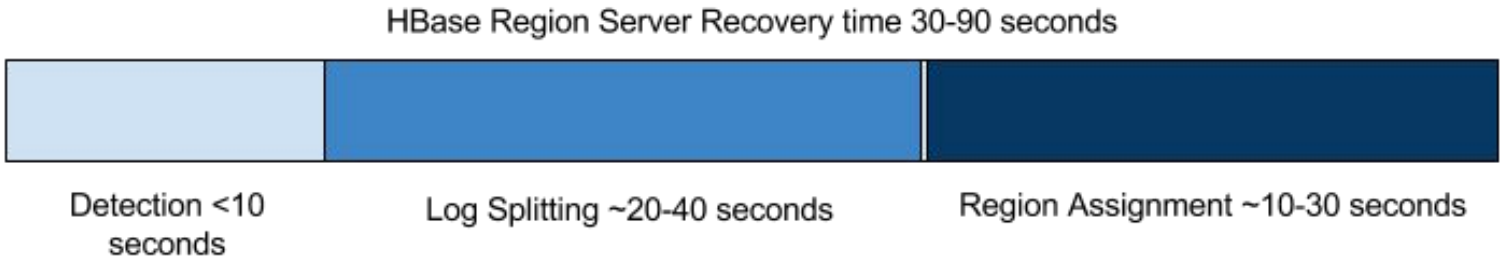
- HubSpot’s Big Data Team provides HBase as a Service to our **50+ Product and Engineering teams.**
- HBase is Hubspot’s canonical Data store and reliably stores all of our customers’ data

What problems we're we solving

- In a typical month we **lose ~3.5% of our Region Servers** to total network failure
- **3500+ microservices interacting with 350+ tables**
- 500+ TBs of raw data served from HBase
- We run an internally patched version of CDH 5.9.0
- We've reached **99.99% availability in every cluster.**

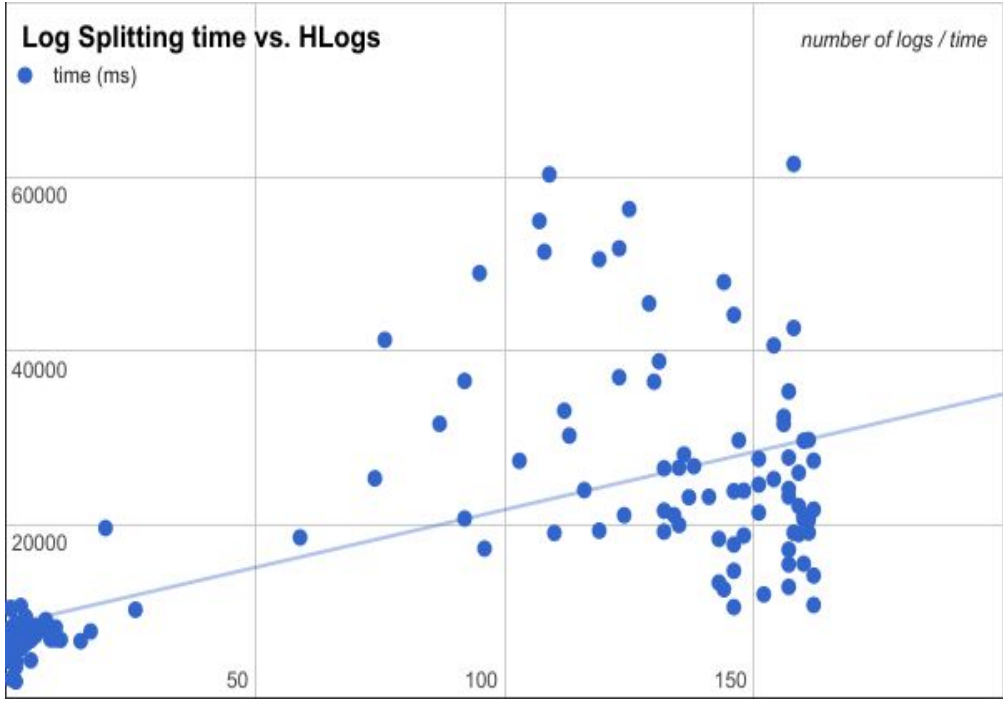
Reducing the cost of failure

What does a Region Server crash look like?



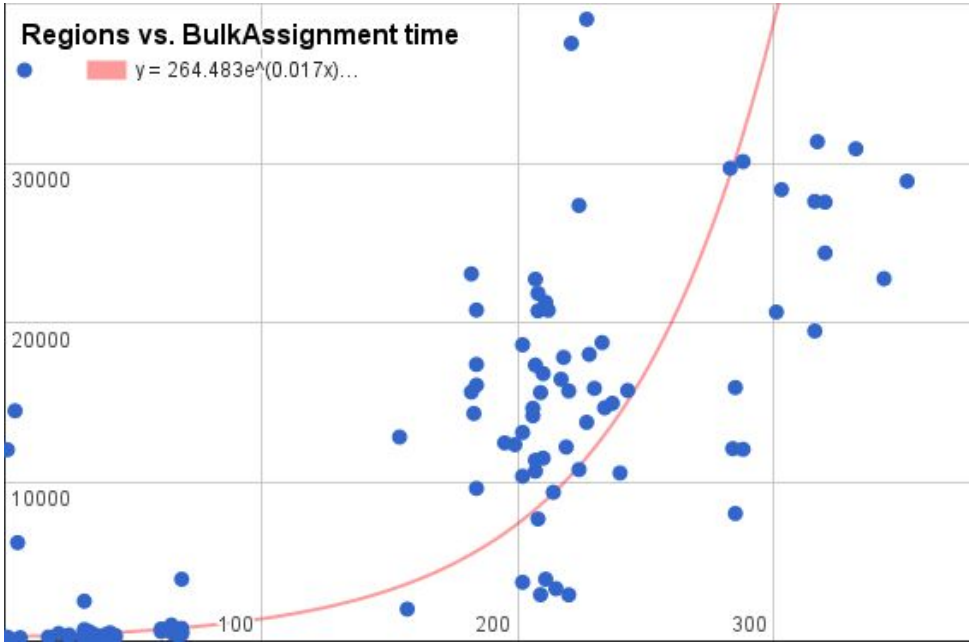
Distributed log splitting performance

- Minimum of 5 seconds
- Scales linearly with the number of HLog files
- **Performance grows erratic with the number of HLogs**



Region assignment & log replay

- **This Process is Non-Linear**
- Scales at least by Regions * HLogs



Improving mean time to recovery (MTTR)

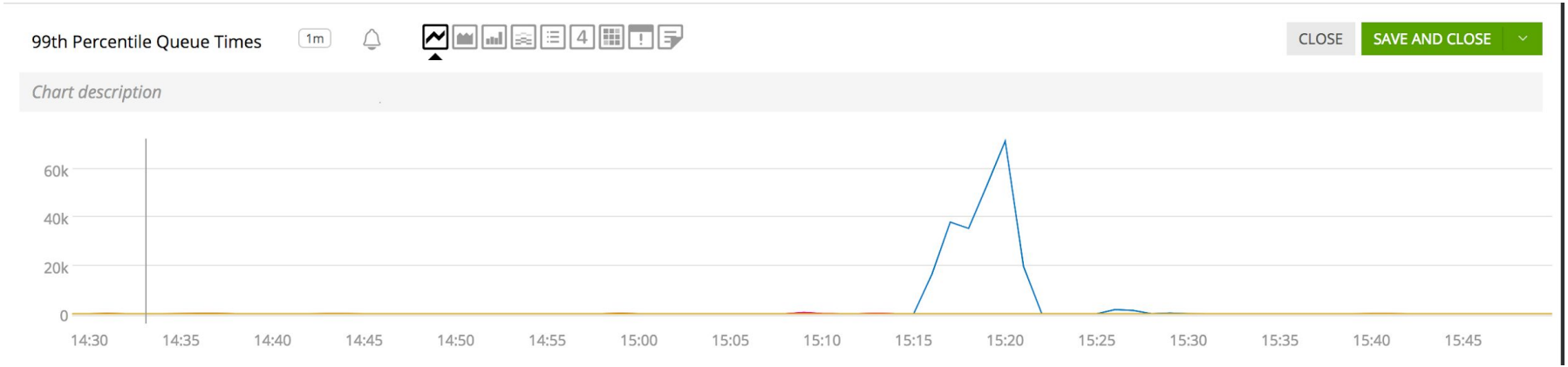
- **Running more and smaller instances** can improve overall availability
- Migrating from d2.8xls to d2.2xls reduced our MTTR from **90 seconds to 30 seconds**

d2.8xl	d2.2xl
244GB ram	61GB ram
24*2TB	7*2TB HDD
36 cores	8 cores

… But the AsyncProcess can get in the way

- AsyncProcess & AsyncRequestFutureImpl didn’t respect operation timeout
- Meta fetches didn’t respect operation timeout
- Open connections to an offline meta server could cause the client to block until socket failure far into the future.

What about stalled instances?



What is the cause?

- Degraded hardware performance?
  - Slow I/O
  - Stuck kernel threads
  - High packet loss
- Long GC pauses?
- Hot regions?
- Noisy user?

Strategy

- Eliminate usage related failures
- Tune GC bit.ly/hbasegc
- Monitor and proactively remove misbehaving hardware

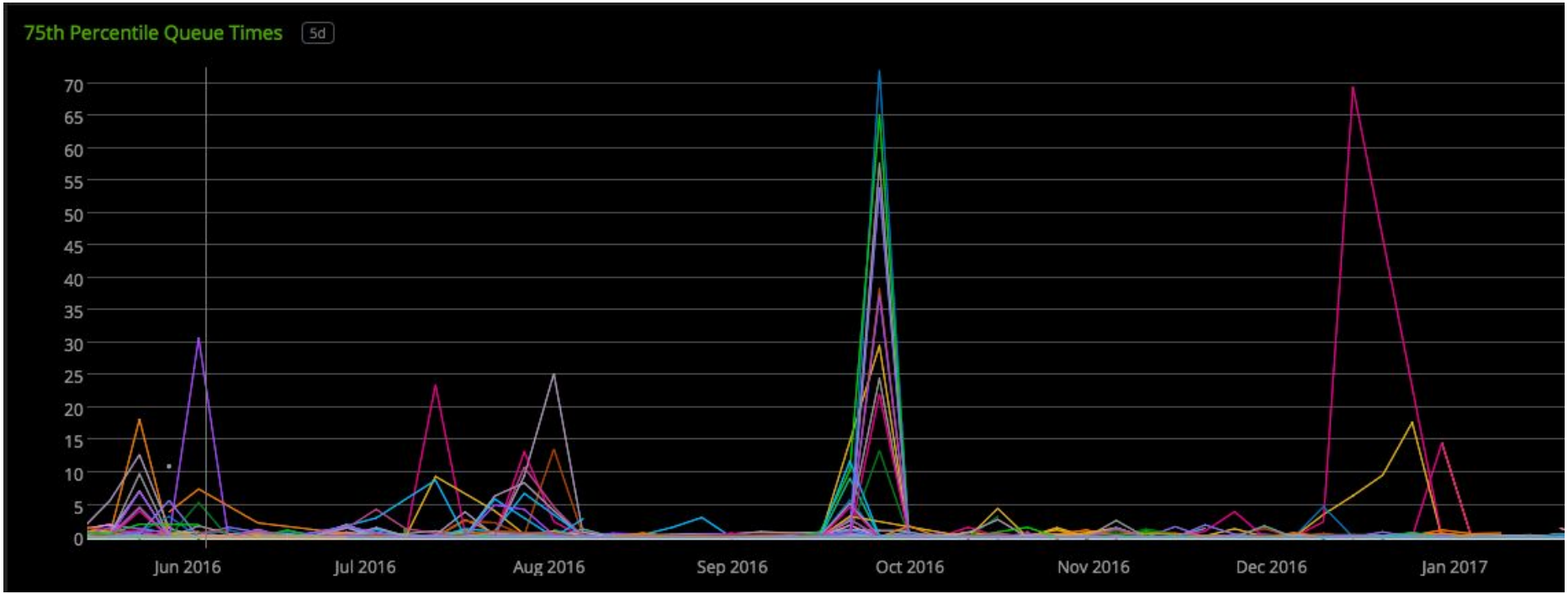
Eliminating usage failures

---

1. Keep regions less than 30GB
2. Use the Normalizer to ensure tables do not have unevenly sized regions
3. Optimize the balancer for multi-tenancy
4. Usage limits and Guardrails

Improving the balancer

Problem: HBase is ~~*unstable*~~, *unbalanced*



We investigated and found one issue...

---

1. Balancer stops if servers host same number of regions

We investigated and found ~~one~~ two issues...

---

1. Balancer stops if servers host same number of regions

2. Requests for tables aren’t distributed across cluster

We investigated and found ~~one~~ ~~two~~ three issues...

---

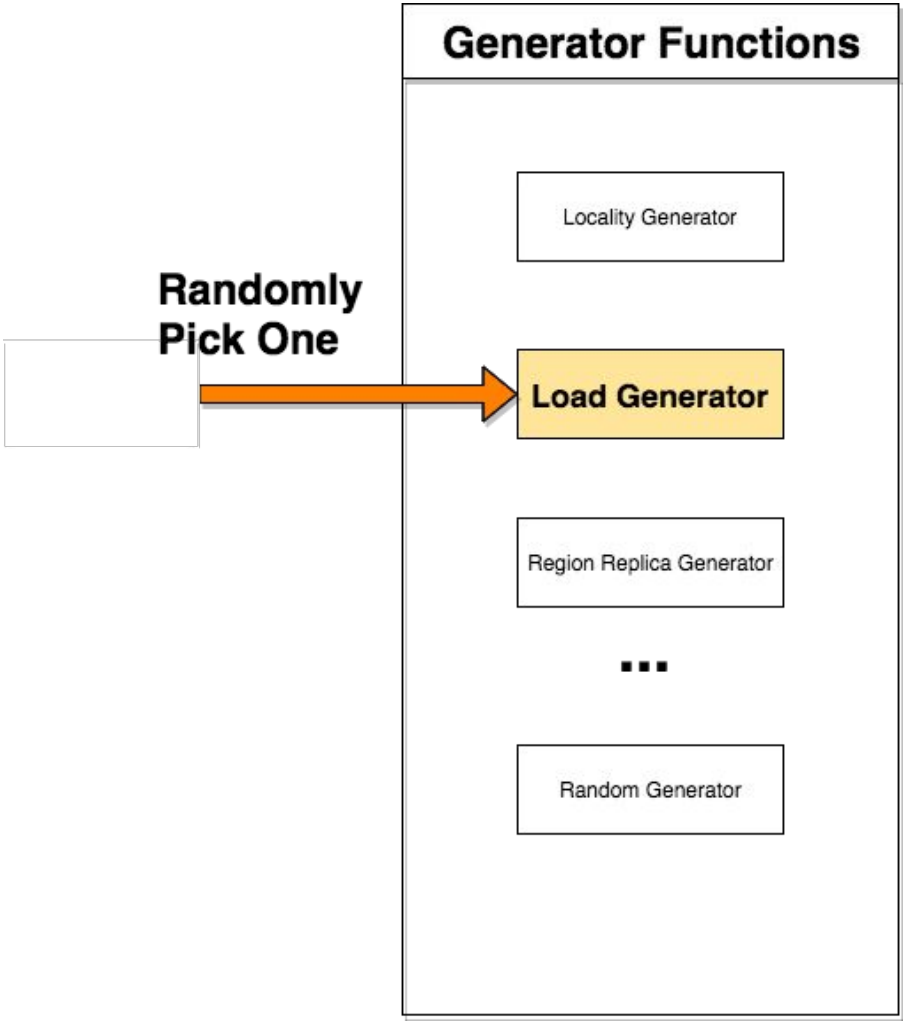
1. Balancer stops if servers host same number of regions

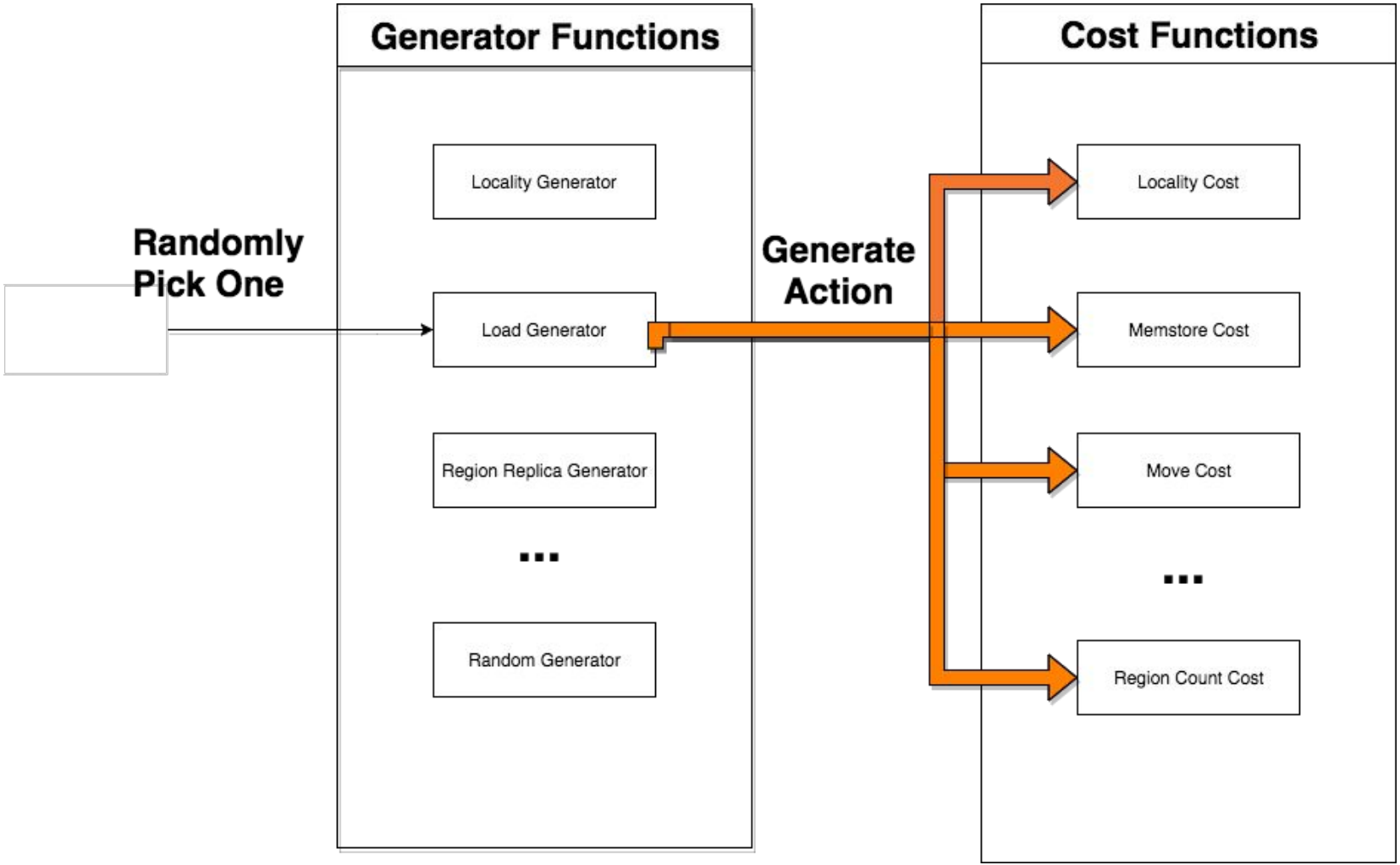
2. Requests for tables aren’t distributed across cluster

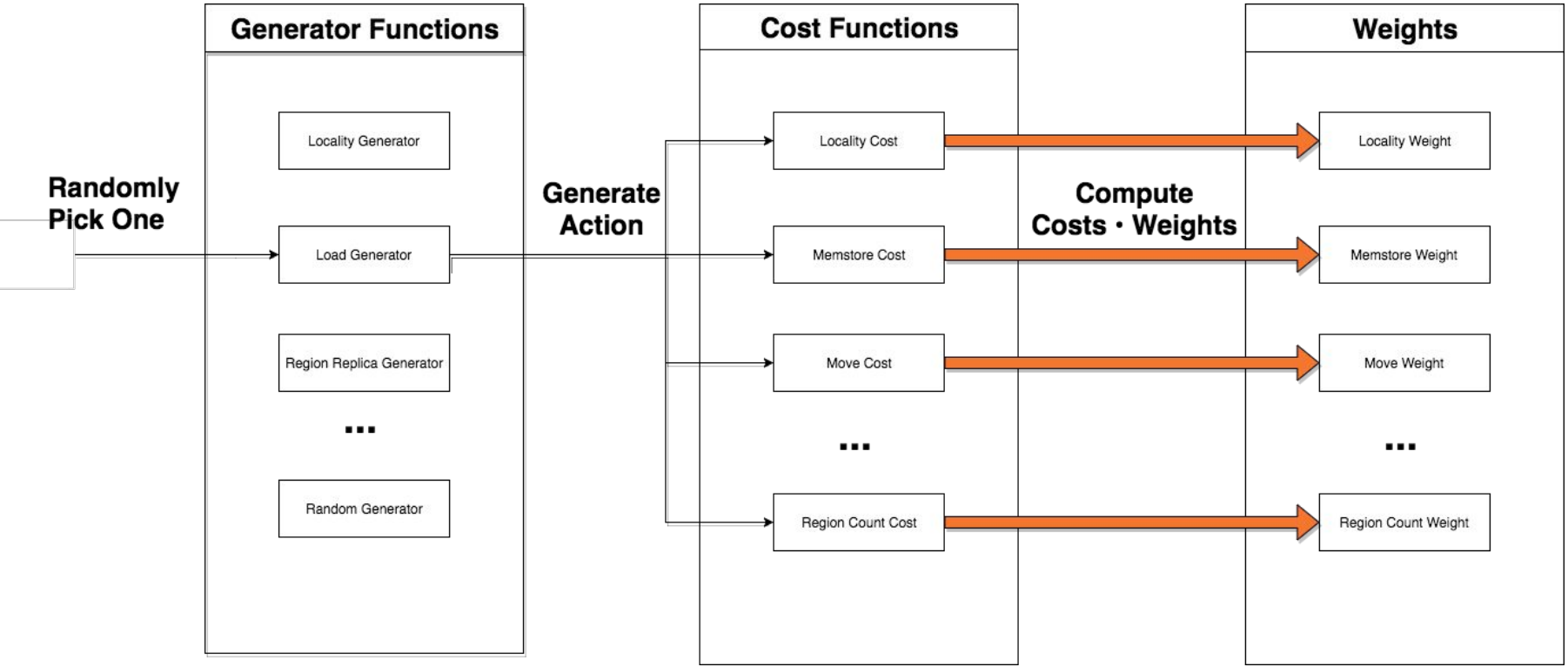
3. Balancer performance doesn’t scale well with cluster size

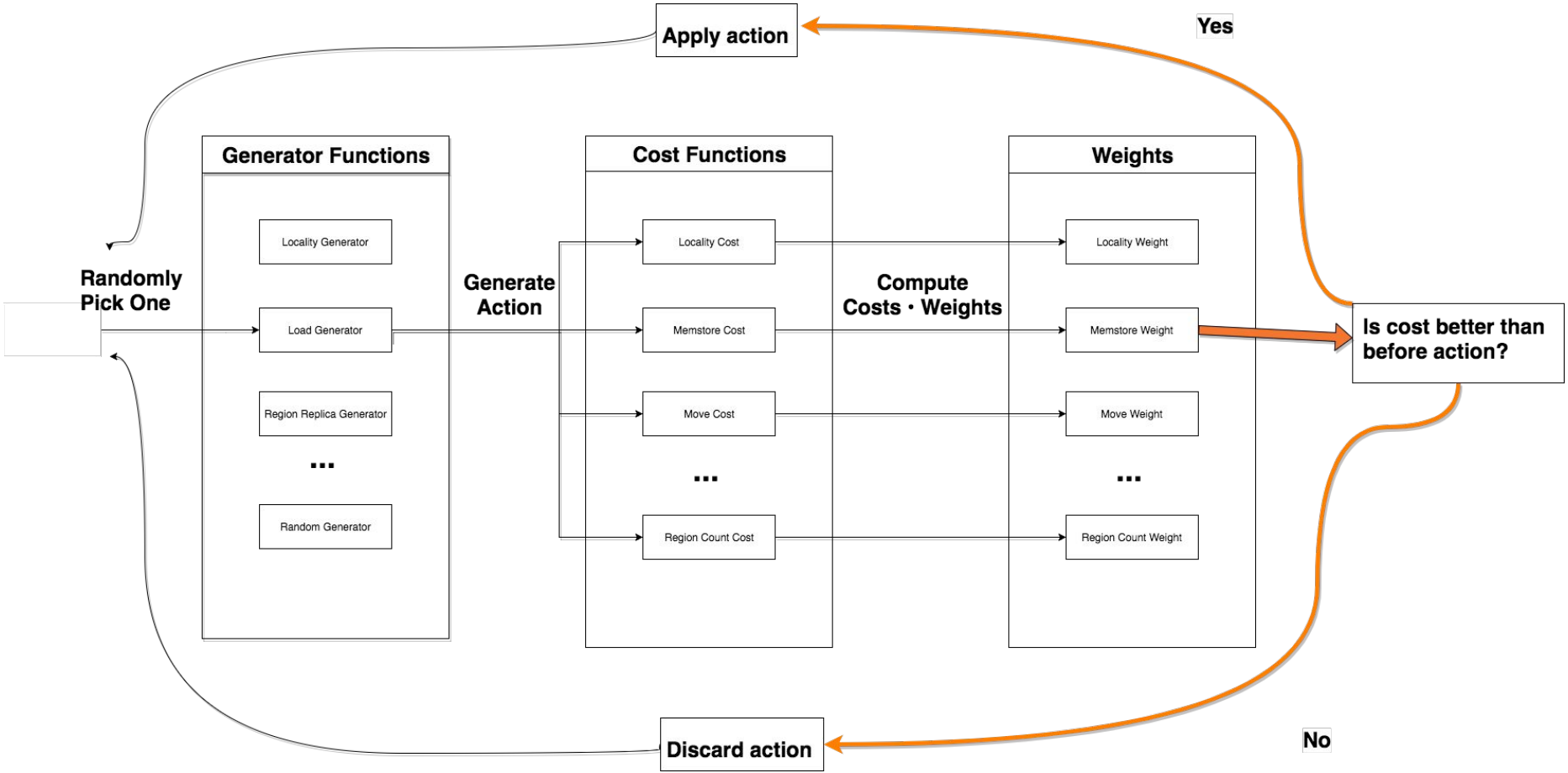
Quick refresher on stochastic load balancer

There will be pictures...


Generator Functions
Locality Generator
Randomly
Pick One
Load Generator
Region Replica Generator
...
Random Generator


Generator Functions
Cost Functions
Locality Generator
Randomly
Pick One
Load Generator
Generate
Action
Region Replica Generator
...
Random Generator
Locality Cost
Memstore Cost
Move Cost
...
Region Count Cost


Generator Functions
Locality Generator
Load Generator
Region Replica Generator
...
Random Generator
Randomly Pick One
Generate Action
Cost Functions
Locality Cost
Memstore Cost
Move Cost
...
Region Count Cost
Compute Costs · Weights
Weights
Locality Weight
Memstore Weight
Move Weight
...
Region Count Weight


Apply action
Yes
Generator Functions
Cost Functions
Weights
Locality Generator
Locality Cost
Locality Weight
Randomly Pick One
Load Generator
Generate Action
Memstore Cost
Compute Costs · Weights
Memstore Weight
Is cost better than before action?
Region Replica Generator
Move Cost
Move Weight
...
...
...
Random Generator
Region Count Cost
Region Count Weight
Discard action
No

The first issue

---

1. Balancer stops if servers host same number of regions

The first ~~issue~~...turned out to be intentional

---

1. Balancer stops if servers host same number of regions

Theory: design choice when balancer was simpler

So our solution is perhaps a design choice

---

1. ~~Balancer stops if servers host same number of regions~~

Theory: design choice when balancer was simpler

Solution: disable the “slop” logic for the stochastic load balancer

The second issue

---

2. Requests for tables aren’t distributed across cluster

The second issue, involved some theorizing

---

2. Requests for tables aren’t distributed across cluster

Theory: cluster has high *table skew*

But every ~~good~~ theory requires a bit of background...

– – –

What is table skew?

Balanced:

	Table 1				Table 2		
Server 1	R1	R2	R3	R4	Q1	Q2	Q3
Server 2	R5	R6	R7	R8	Q4	Q5	
Server 3	R9	R10	R11	R12	Q6	Q7	Q8
Server 4	R13	R14	R15	R16	Q9	Q10	Q11

Unbalanced:

	Table 1							Table 2				
Server 1	R1	R2	R3	R4	R5	R6	R7	Q1	Q2	Q3	Q4	Q5
Server 2	R8	R9	R10	R11				Q6				
Server 3	R12	R13						Q7	Q5	Q6		
Server 4	R14	R15	R16					Q8	Q9	Q10	Q11	

What is table skew?

Balanced:

	Table 1				Table 2		
Server 1	R1	R2	R3	R4	Q1	Q2	Q3
Server 2	R5	R6	R7	R8	Q4	Q5	
Server 3	R9	R10	R11	R12	Q6	Q7	Q8
Server 4	R13	R14	R15	R16	Q9	Q10	Q11

Unbalanced:

	Table 1							Table 2				
Server 1	R1	R2	R3	R4	R5	R6	R7	Q1	Q2	Q3	Q4	Q5
Server 2	R8	R9	R10	R11				Q6				
Server 3	R12	R13						Q7	Q5	Q6		
Server 4	R14	R15	R16					Q8	Q9	Q10	Q11	

Problem: When *unbalanced*, all requests to `R1-R7` and `Q1-Q5` hit `Server 1`

If cluster has high table skew, we expect...

- High table skew cost

- High responsiveness to tuning table skew weight

But we found...

- ~~High~~ **low** table skew cost

- ~~High~~ **low** responsiveness to tuning table skew weight

So we dug in and discovered...

- Regions are **not** evenly distributed across the cluster

- Table skew **is** present, **contrary to what balancer reports**

Time for a new theory...

---

2. Requests for single tables hit subset of region servers

Theory: *table skew* not properly computed

And a solution!

~~2. Requests for single tables hit subset of region servers~~

Theory: *table skew* not properly computed

Solution: Create new table skew cost function and generator

New table skew cost function (HBASE-17707)

- Computes number of moves/swaps required to evenly distribute each table’s regions

New table skew cost function (HBASE-17707)

- Computes number of moves/swaps required to evenly distribute each table’s regions

- Outputs a value between [0, 1] proportional to the number of moves required

New table skew cost function (HBASE-17707)

- Computes number of moves/swaps required to evenly distribute each table’s regions

- Outputs a value between [0, 1] proportional to the number of moves required

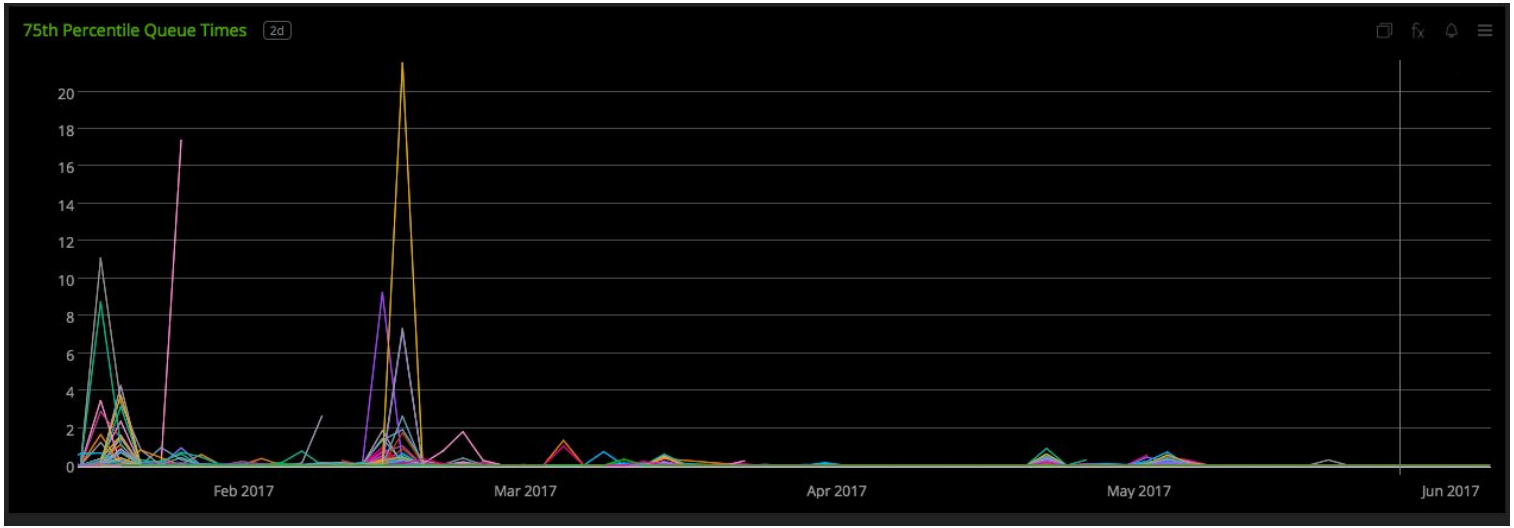
- Analogous to “edit distance” between two words

New table skew candidate generator (HBASE-17707)

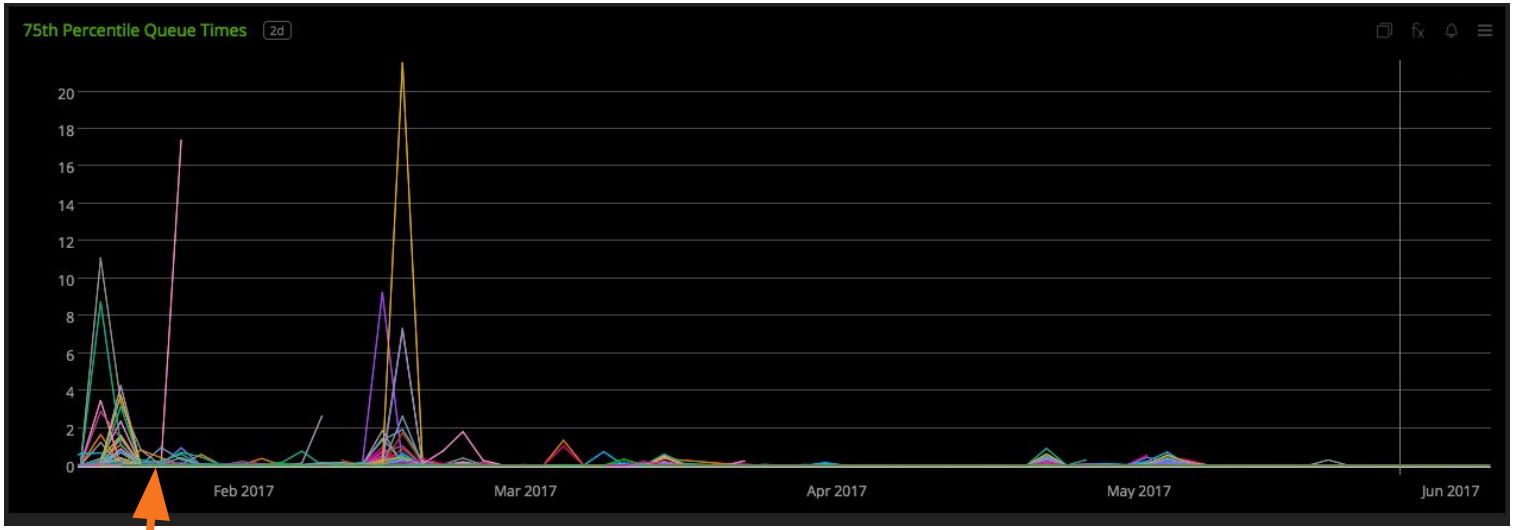
Propose region moves/swaps to optimize TableSkewCostFunction

How did our fix work in practice?

75th Percentile Request Queue Times



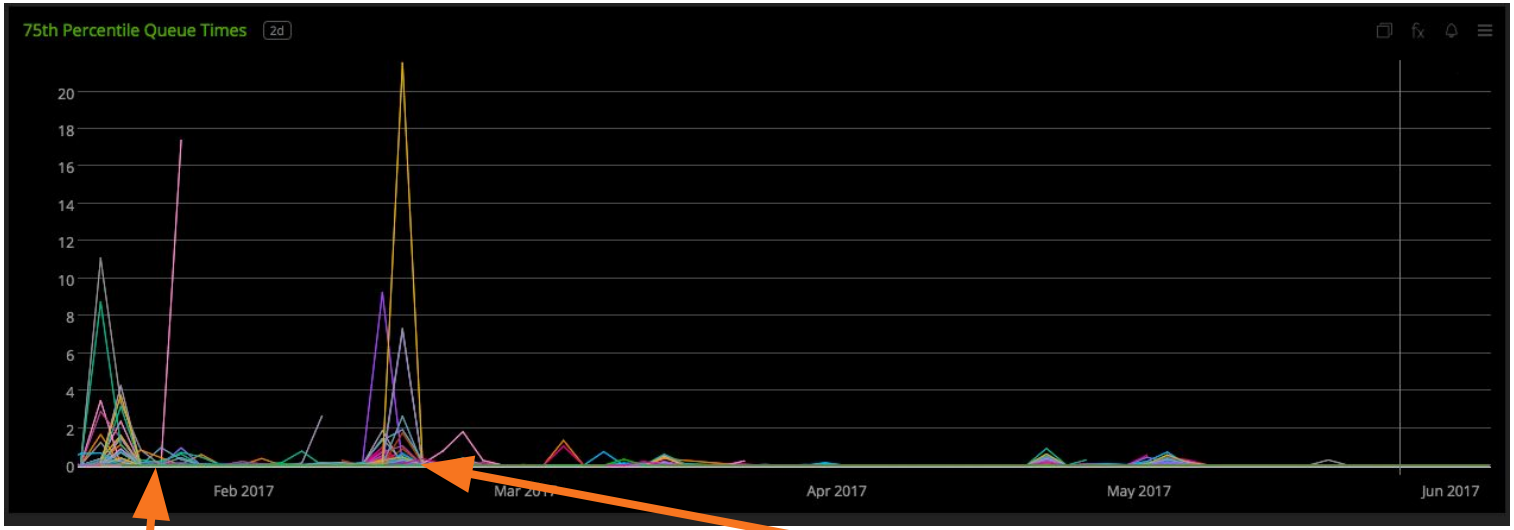
Spot the deploy




```
commit 88f288fc405ed9baf66172e41ddccd19acc32bd3
Author: Kahlil Oppenheimer <koppenheimer@hubspot.com>
Date:   Wed Jan 25 14:29:12 2017 -0500

    Added new TableSkew[CostFunction/CandidateGenerator]
```

Spot the deploy (and the bug fix)




```
commit 88f288fc405ed9baf66172e41ddccd19acc32bd3
Author: Kahlil Oppenheimer <koppenheimer@hubspot.com>
Date:   Wed Jan 25 14:29:12 2017 -0500

    Added new TableSkew[CostFunction/CandidateGenerator]
```

```
commit c8fe0b17e93c629a572d9ffc63ea58df1598b7a4
Author: Kahlil Oppenheimer <koppenheimer@hubspot.com>
Date:   Wed Feb 15 17:18:40 2017 -0500

    Improved server/table selection for table skew
```

Back to investigating...

---

3. Balancer performance doesn’t scale well with cluster size

Back to investigating...

---

3. Balancer performance doesn’t scale well with cluster size

Theory: table skew calculation takes too long

Back to investigating...

---

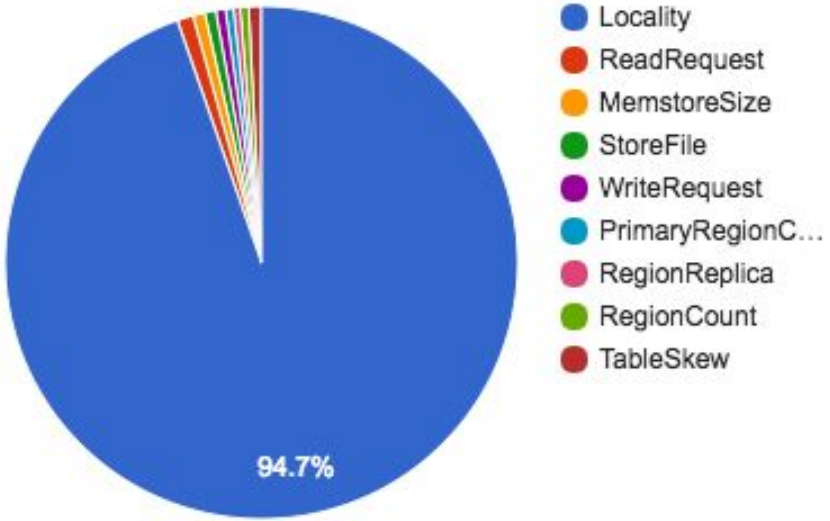
3. Balancer performance doesn’t scale well with cluster size

~~Theory: table skew calculation takes too long~~

But new table skew code didn’t improve performance...

So we benchmarked cost functions

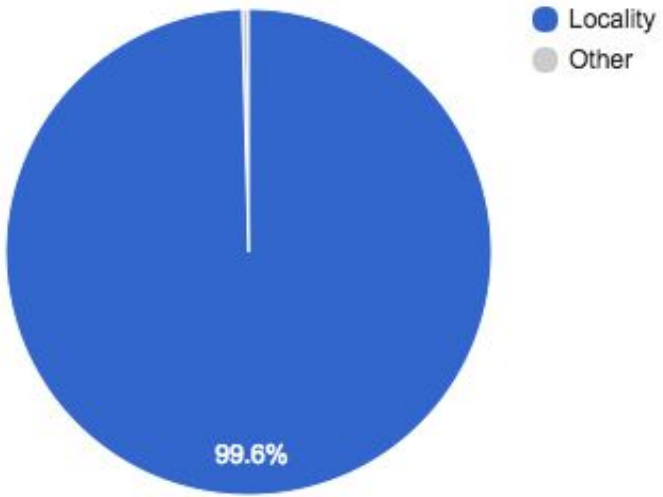
But found a surprise



We benchmarked generators too...

– – –

And...surprise!



What is locality?

A measure of how close the data is to the computation

For HBase: How much of a region’s data is on its host’s disk

Back to theorizing

---

3. Balancer performance doesn’t scale well with cluster size

~~Theory: table skew calculation takes too long~~

Theory: *locality* computation takes too long

And solution-izing

---

3. Balancer performance doesn’t scale well with cluster size

Theory: *Locality* computation takes too long

Solution: Redesign locality cost function and generator

The old locality code

- Recomputes locality of every region on every server

The old locality code

- Recomputes locality of every region on every server

- Reads every HDFS block location of every region

The old locality code

- Recomputes locality of every region on every server
- Reads every HDFS block location of every region
- **For every action the balancer proposes!**

New locality code (HBASE-18164)

- Caches localities of all regions on all servers

New locality code (HBASE-18164)

- Caches localities of all regions on all servers

- Incrementally updates locality cost based on proposed region moves/swaps

New locality code (HBASE-18164)

- Caches localities of all regions on all servers

- Incrementally updates locality cost based on proposed region moves/swaps

- Add per-rack locality computation
  - **Reduce AWS charges** for inter-rack data transfer

But does it work?

– – –

But ~~does it~~ **it does** work

- **8-10x** more balancer actions considered **in all clusters**
  - **20x in big clusters**

But ~~does it~~ **it does** work

- **8-10x** more balancer actions considered **in all clusters**
  - **20x in big clusters**
- LocalityCandidateGenerator from 30,485ms to 4,721ms
  - **~6x faster**

But ~~does it~~ **it does** work

- **8-10x** more balancer actions considered **in all clusters**
  - **20x in big clusters**
- `LocalityCandidateGenerator` from 30,485ms to 4,721ms
  - **~6x faster**
- `LocalityCostFunction` from 28,267ms to 57ms
  - **~495x faster**

Balancer changes summarized

---

1. Balancer is smarter about table skew

2. Balancer has more time to “think” with less time spent in locality computation

3. Cluster stability increased with better balance

Eliminating workload driven failure

Limit RPC handler usage by service

- We use a semaphore to limit the number of active CallRunners any given hadoop User can have open
- In the event that insufficient RPC handlers are available **to a hadoop user** return a Retryable Exception and rely on client side exponential backoff

Client guardrails

- With 3500+ services communicating best practices is difficult
- **Communicate best practices with code!**

Max Request Size	10 MB
Max Response Size	10 MB
Max Batch Size	20,000
Max Read Columns	20,000
Max Write Cells	20,000

Reducing the impact of hardware failure

Prioritize removal of bad hardware

- Watch kernel logs for
  - I/O failures
  - Stuck Threads
  - General kernel errors
- Prioritize removal of these instances to prevent regionserver stalls
- We use an in-house orchestration tool known as Tranquility for this

Now we'll take questions :)

Thanks for listening. Feel free to reach out!

jcmoore@hubspot.com, kahlil@hubspot.com