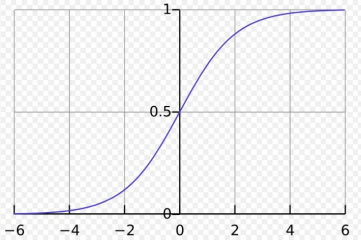
Gemini Storm Use Cases

07-13-2016

Query-Ad Clickability (Refresh memory)



- Compute clickability for a given [Query, Ad] pair at runtime in AdServer
- Logistic regression model - computed offline

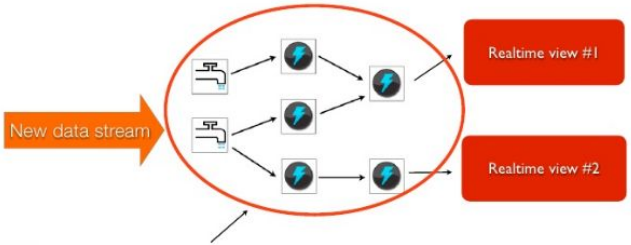
$$p = \frac{1}{1 + e^{-(b_0 + b_1 x_1 + b_2 x_2 + \cdots + b_p x_p)}}$$

  - Query-Ad Textual Features
  - Historical aggregation metrics (ec/coec) at different levels
    - Query, Query-Ad, Query-Campaign etc
    - Computed offline based on last 6 months data (moving window)
      - Available at run time to get ec/coec values for a given feature key (NCTR servers)
  - Model contains features based on ec/coec values (ec/coec values are binned)
    - Example: if Query level ec between [a, b] and coec between [d, e], then weight = b2
    - MOBILE^Q_EC=3.721758__Q_COEC=1.896969 0.067093
- Added real time features (last x hours) based on [User, Query, Ad] ec/coec
  - Compute ec/coec values for a given [user, query, ad] based on last x hours imps

Near Real Time Click Feedback

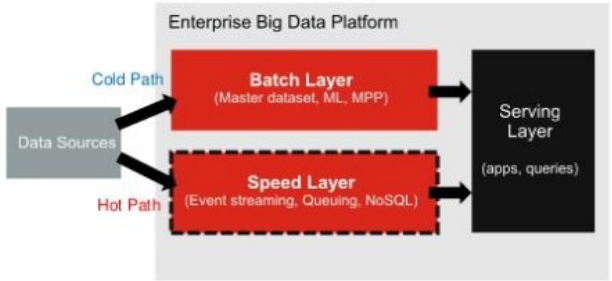
- Get north impressions/clicks for a given user (bcookie) in the last x interval
- Derive ec/coec metrics at User, [User, Query], [User, Ad Domain] using real time user level north impressions/clicks
- Use user real-time features in addition to other long term features for a given [user, query, ad] to predict clkb more accurately

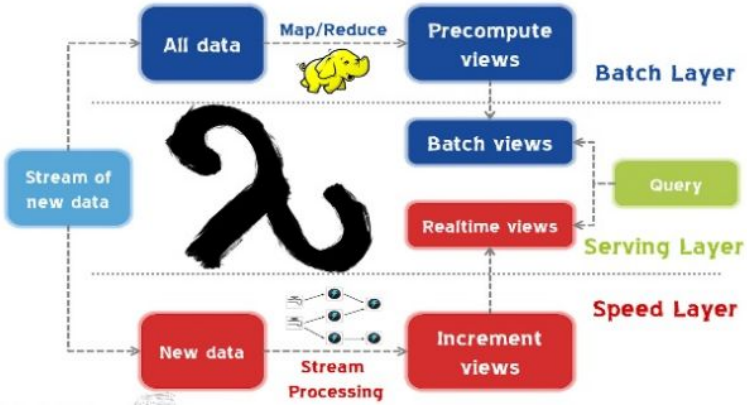
What is Storm?



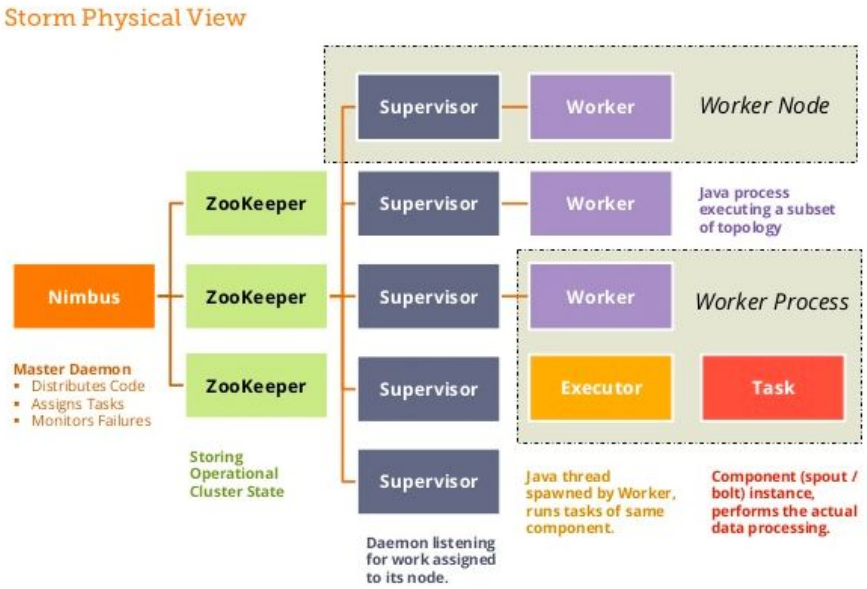
- Distributed and fault-tolerant real-time computation
  - Streaming
  - Scalable (Thousands of workers per cluster)
  - Fault Tolerant (Failure is expected but embraced)
  - Reliable (Guaranteed message delivery)
  - Key component in Lambda Architecture

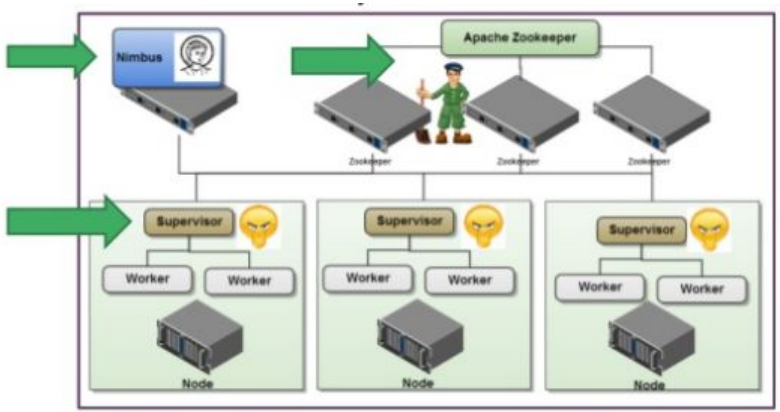
Storm



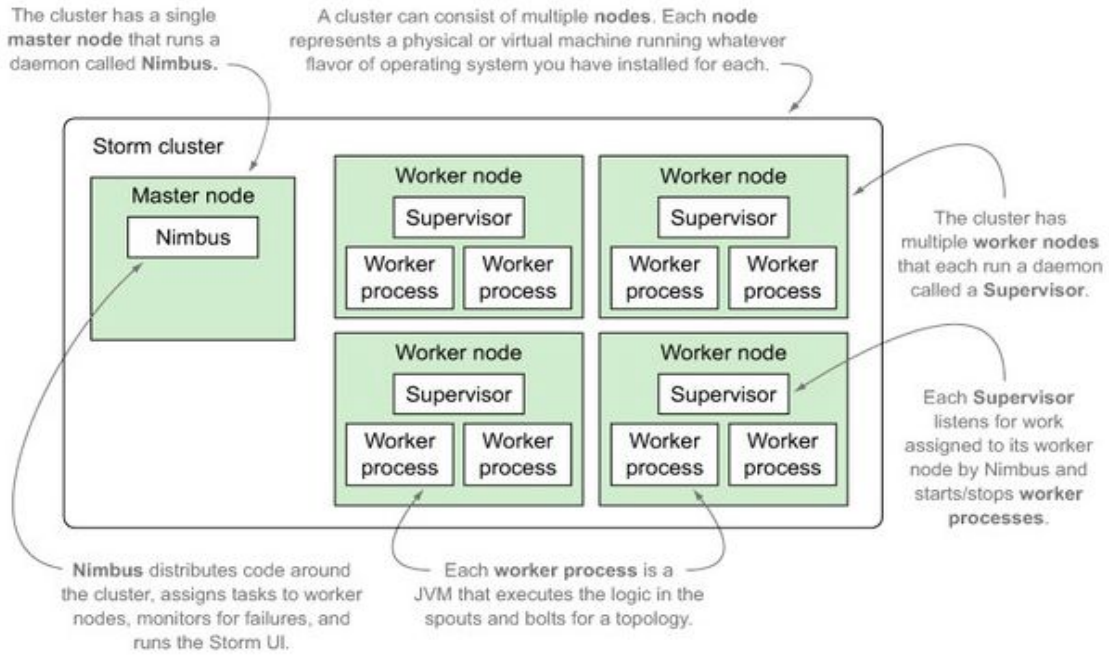


Storm Cluster Physical View





Storm Cluster



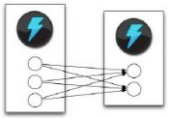
Storm Core Concepts

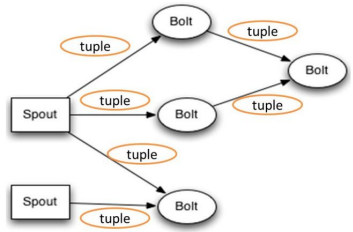
- Tuple
  - Named list of values

- Stream
  - Unbounded sequence of Tuples

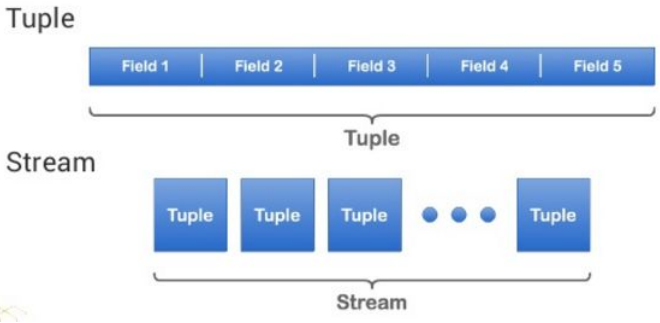
- Spout/Bolts have interfaces to implement to run application specific logic
  - Spout - Source of Streams

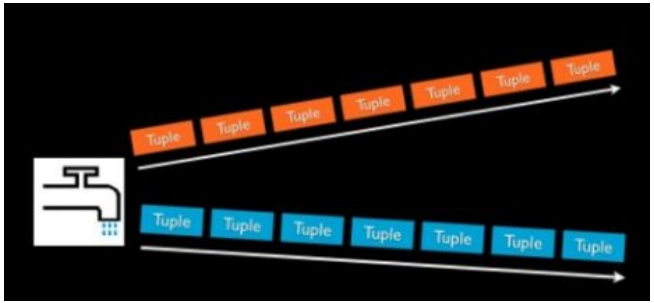
  - Bolt - Consume and process any number of input streams and emit new streams
- Topology
  - Graph of computation. Each node (spout/bolt) in a topology contains processing logic
  - Links between nodes indicate how data should be passed around between nodes
- Stream Grouping
  - How to send tuples between two components
  - When a tuple is emitted, to which task does it go to ?



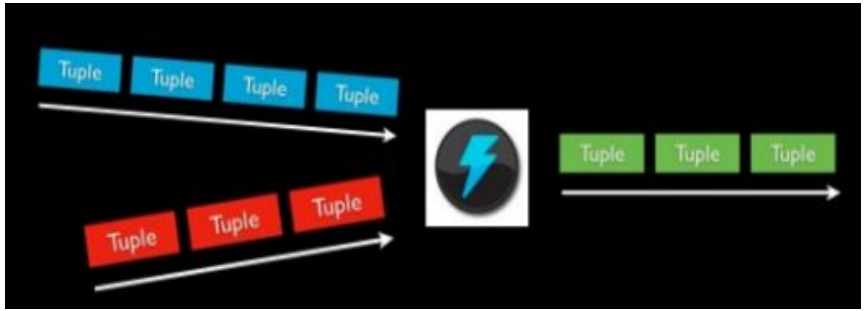


Tuple, Stream, Spout, Bolt and Topology

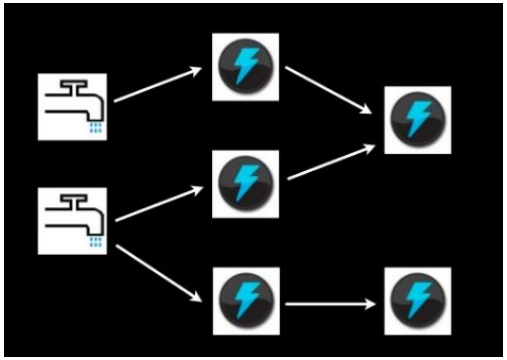




Spout: Source of Streams (Ex: Read from Kafka Topic)



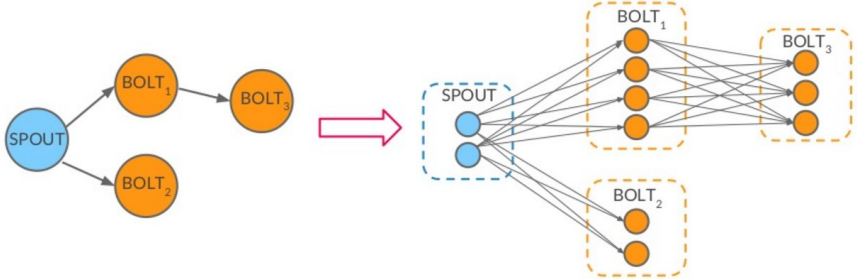
Bolt: Processes input streams and produces new streams
Ex: Filters, Aggregation, Joins, Persist to DB

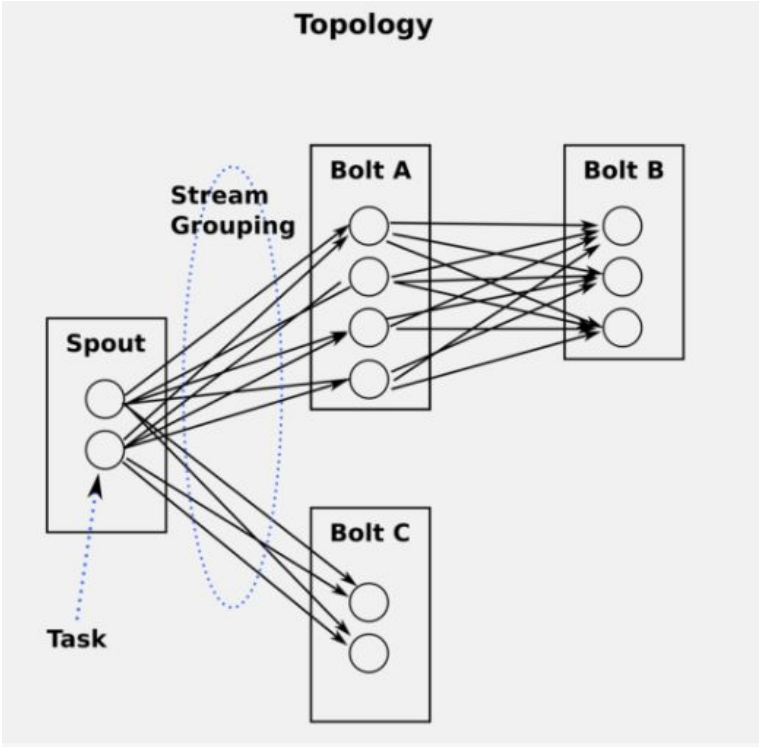


Topology: DAG of spouts and Bolts

Topology Deployment

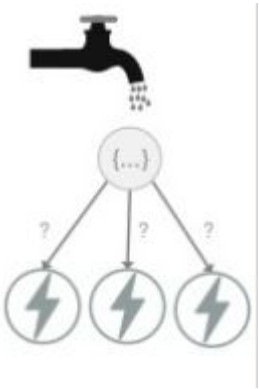
- Topology
  - Spouts + Bolts + Tuples + Streams
  - A network of spouts and bolts wired together into a workflow
  - Parallelism is achieved by running multiple replicas of the same spout/bolt
- Tasks
  - Multiple instances of Components (spout or bolt) to perform computational tasks



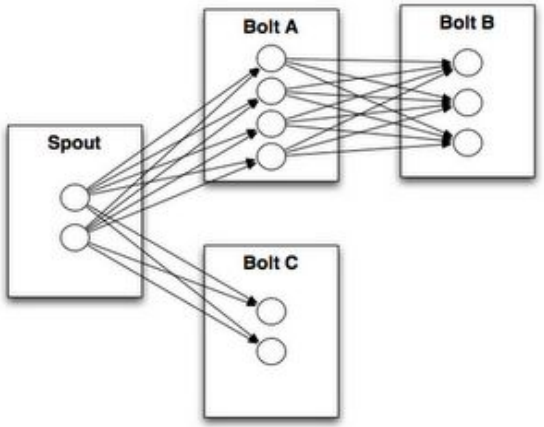


Stream Grouping

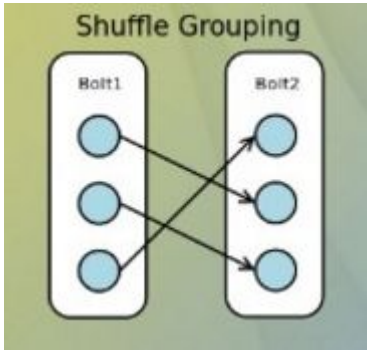
- Determines how storm routes tuples between tasks in topology

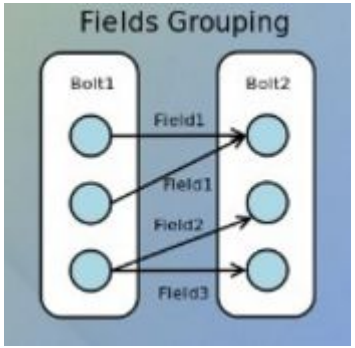


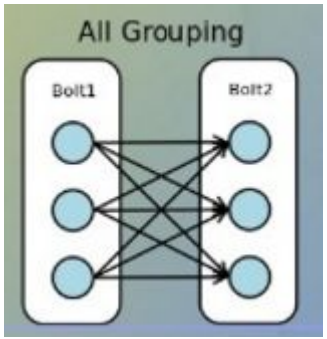
Stream Grouping

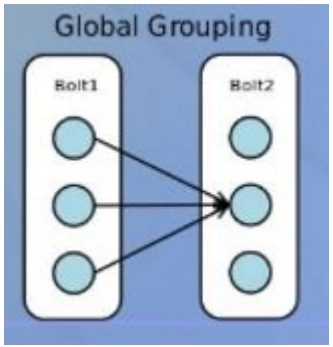


- Data is shuffled from a producer (spout/bolt) to bolt
- Partitioning strategies
  - ShuffleGrouping - Randomly partitions tuples
  - FieldsGrouping - Hashes on a subset of tuple attributes/fields
  - AllGrouping - Replicates entire stream to all the consumer tasks
  - GlobalGrouping - Sends entire stream to a single bolt
  - LocalGrouping - Send tuples to the consumer bolts in the same executor



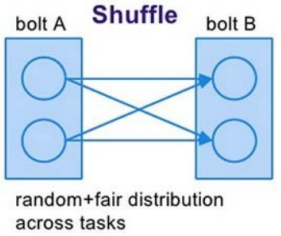


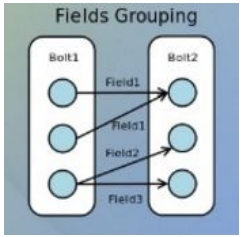




Shuffle Grouping Vs Fields Grouping

- Shuffle Grouping
  - Tuples are randomly distributed across bolt's tasks so that each bolt is guaranteed to get equal number of tuples
- Fields Grouping
  - The stream is partitioned by the fields specified in the grouping
  - For example, if the stream is grouped by the "user-id" field, tuples with the same "user-id" will always go the same task, but tuples with different "user-id" may go to different tasks





Spout, Bolt, Bolt Output API

Spout API

```
public interface ISpout extends Serializable {

    void open(Map conf,
              TopologyContext context,
              SpoutOutputCollector collector);

    void close();

    void activate();

    void deactivate();

    void nextTuple();

    void ack(Object msgId);

    void fail(Object msgId);
}
```

Core API

Bolt API

```
public interface IBolt extends Serializable {

    void prepare(Map stormConf,
                 TopologyContext context,
                 OutputCollector collector);

    void cleanup();

    void execute(Tuple input);
}
```

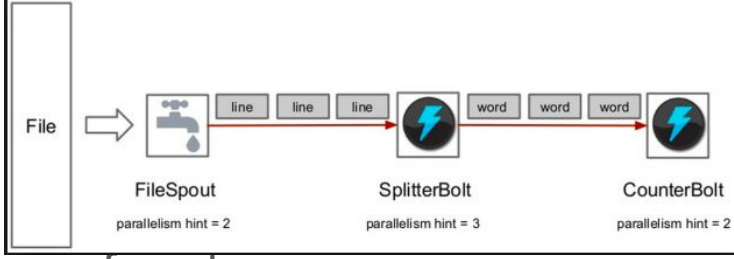
Core API

Bolt Output API

```
public interface IOutputCollector extends IErrorReporter {

    List<Integer> emit(String streamId,
                       Collection<Tuple> anchors,
                       List<Object> tuple);

    void emitDirect(int taskId,
                    String streamId,
                    Collection<Tuple> anchors,
                    List<Object> tuple);

    void ack(Tuple input);

    void fail(Tuple input);
}
```

Core API

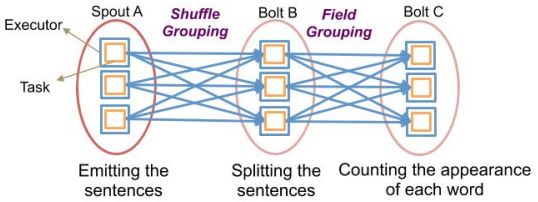
Word Count Storm Topology



- Word Count: Count the different words in a stream of sentences
- Implemented by 3 classes and composed to obtain the desired topology
  - SentenceSpout (FileSpout)
    - Emits a stream of tuples that represent sentences
      - {sentence: “Introduction to storm” }
  - SplitSentenceBolt (SplitterBolt)
    - Emits a tuple for each word in the sentences it receives
      - {word: “Introduction”} {word: “to” } {word: “storm”}
  - WordCountBolt (CounterBolt)
    - Updates the count (and finally save counts to some data store)
- Build Topology
  - Wire spouts and bolts together

```
TopologyBuilder builder = new TopologyBuilder();
builder.setSpout("sentencs-spout", new SentenceSpout());
builder.setBolt("split-bolt", new SplitSentenceBolt())
     .shuffleGrouping("sentences-spout");
builder.setBolt("count-bolt", new WordCountBolt())
     .fieldsGrouping("split-bolt", new Fields("word"));
```

WordCount Spout/Bolt




```java
public class SentenceSpout extends BaseRichSpout {
    private SpoutOutputCollector collector;

    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("sentence"));
    }

    public void open(Map config, TopologyContext context,SpoutOutputCollector collector) {
        this.collector = collector;
    }

    public void nextTuple() {
        //prepare the next sentence S to emit
        this.collector.emit(new Values(S));
        //...
    }
}
```

```java
public class SplitSentenceBolt extends BaseRichBolt{
    private OutputCollector collector;

    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("word"));
    }

    public void prepare(Map config, TopologyContext context,OutputCollector collector) {
        this.collector = collector;
    }

    public void execute(Tuple tuple) {
        String sentence = tuple.getStringByField("sentence");
        String[] words = sentence.split(" ");
        for(String word : words){
            this.collector.emit(new Values(word));
        }
    }
}
```

- Each node extends some abstract classes and must implement some basic methods for defining the format of the tuple emitted and business logic.

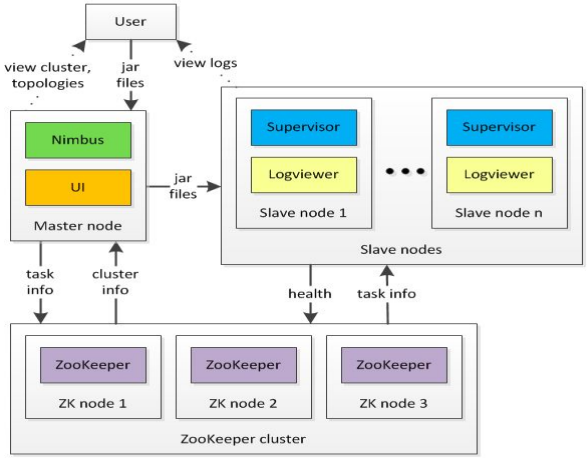
```java
public class WordCountBolt extends BaseRichBolt{
    private OutputCollector collector;
    private HashMap<String,Long> counts=null;

    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        //this bolt does not emit anything
    }

    public void prepare(Map config, TopologyContext context,OutputCollector collector) {
        this.collector = collector;
        this.counts = new HashMap<String,Long>();
    }

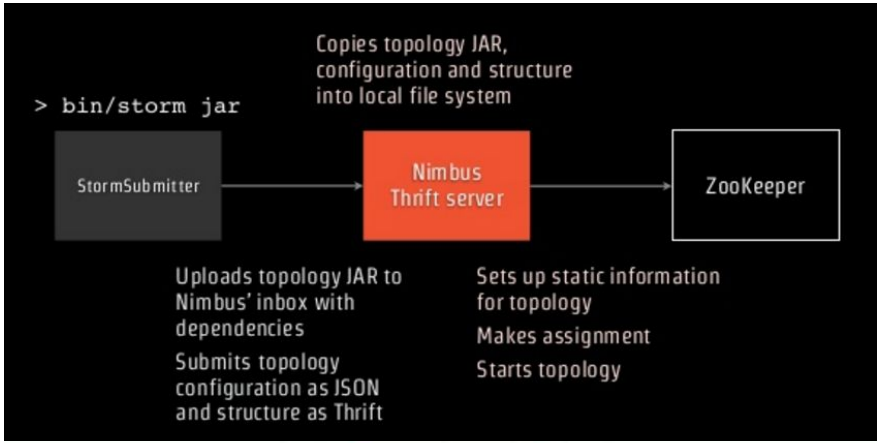
    public void execute(Tuple tuple) {
        String word = tuple.getStringByField("word");
        //...increments count..
    }
}
```

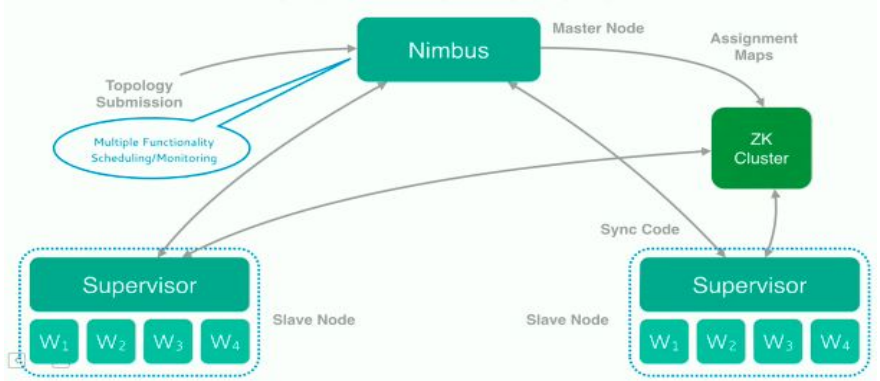
Nimbus Node



- Master node runs a server daemon called nimbus
- Role of nimbus
  - Validate the topology
  - Distribute the topology source code
  - Schedule the topology - Distributes the work among the workers of the cluster
  - Activate the topology - Tells worker nodes to start executing it
  - Monitor the topology - Monitor topology by reading heartbeats sent by worker nodes
- Nimbus commands - Apache thrift service and listens for user commands
  - Submit a topology - using JAR file containing the code
  - Kill topology - can stop running a topology and removes from the cluster
  - Activate/Deactivate topologies
  - Rebalance a topology - can increase/decrease the number of nodes involved in computation

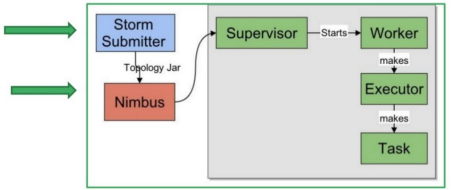
Topology Submission

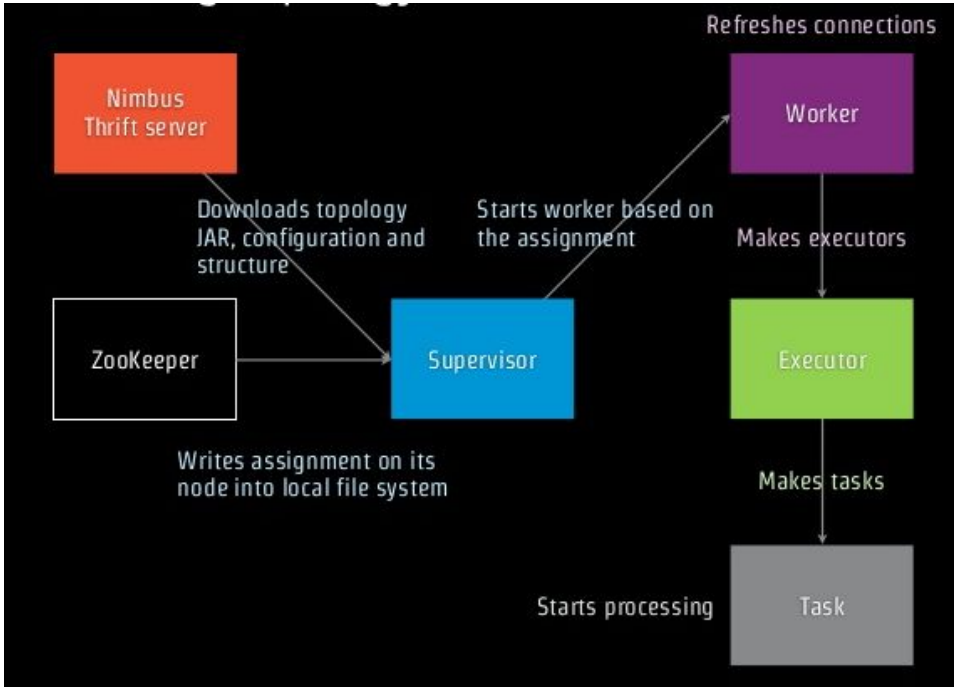




- User uploads topology
- Nimbus calculates assignments and sends to zookeeper
- Supervisor nodes receive assignment information via zookeeper watches
- Supervisor nodes downloads topology from nimbus
- Supervisors spawn workers (JVM process) to start topology

Starting Topology

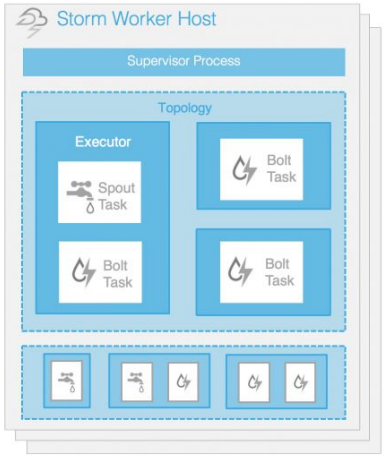


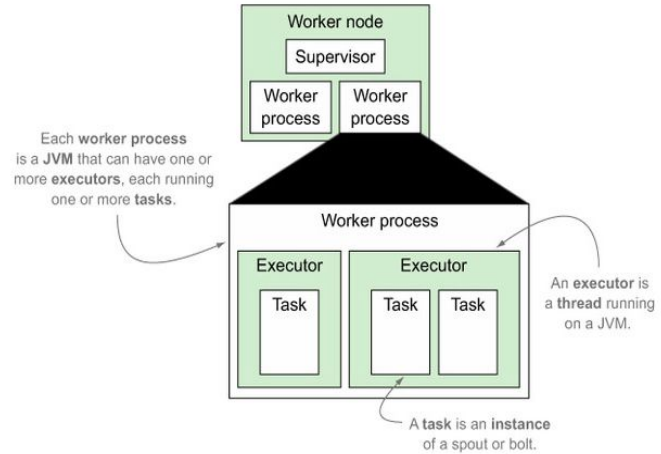


- Supervisor daemon process is listening for work assigned to its node
- Nimbus master daemon process responsible for
  - Code distribution
  - Assigning Tasks
  - Reassignment in case of failures

Anatomy of Worker Node

- Each worker process executes a subset of a topology

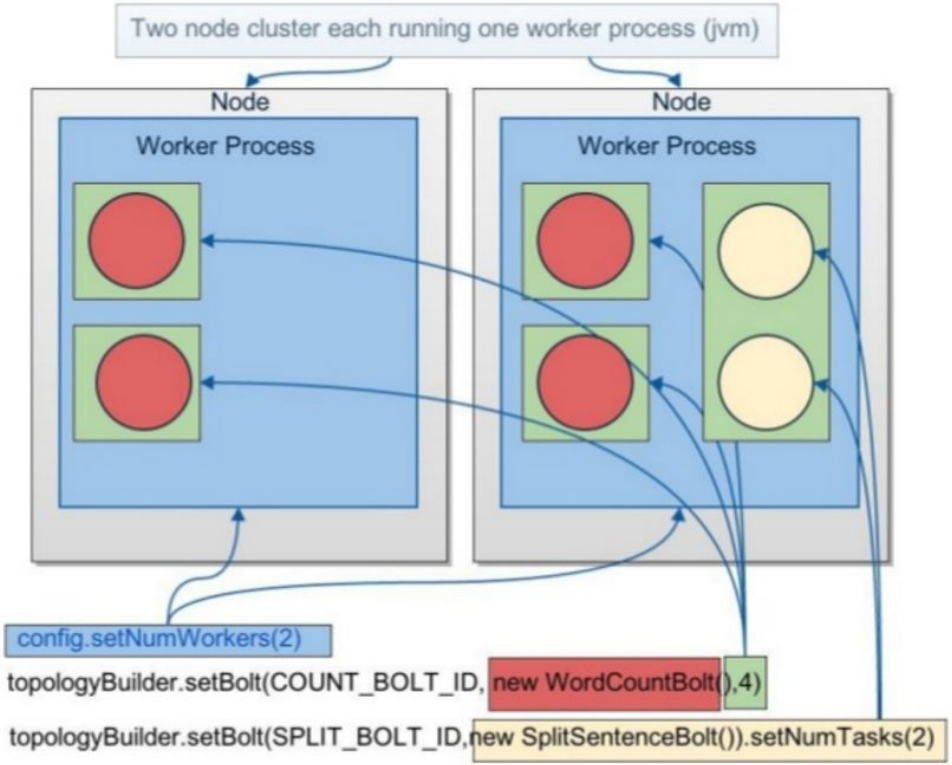




More on Tasks

- Task
  - Performs actual processing and is run with in parent executor’s thread
  - Each spout and bolt executes as many tasks across cluster (many instances of each)
- Tasks per component
  - Number of tasks for a component (Spout/Bolt) is always constant (configured by user in topology) throughout the lifetime of the topology
  - But the number of executors can change over time (rebalance topology)
- By default, the number of tasks is set to be the same as the number of executors (unless user specifies this number in topology)
  - Storm will run one task per thread (default)

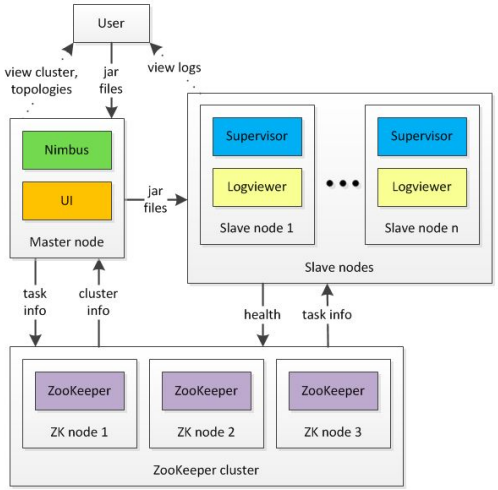
Topology Deployment - Example

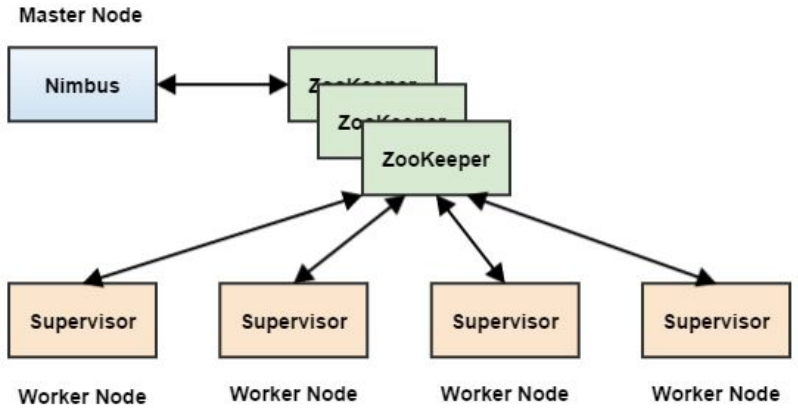


Zookeeper Nodes

```
ls /
[mobileApp, transactional, storm, nrtcf, zookeeper]
ls /storm
[fubariteblue-ni]
ls /storm/fubariteblue-ni
[backpressure, workerbeats, supervisors, errors, logconfigs, profilerconfigs, credentials, storms, assignments]
```

- Coordinates the communication between Nimbus and worker nodes
- State of the storm cluster is stored in Zookeeper
  - Supervisor details
  - Information about the topologies, topology assignments
  - Heartbeats sent by the worker nodes to be read by Nimbus



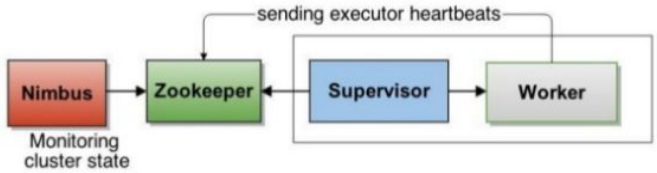


Zookeeper State

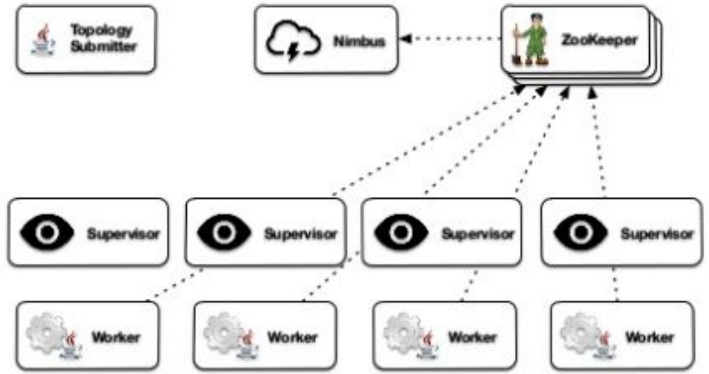
```
./zkCli.sh -server fubariteblue-ni-zk0.blue.ygrid.yahoo.com:50512
ls  /storm/fubariteblue-ni/supervisors
ls /storm/fubariteblue-ni/storms
ls /storm/fubariteblue-ni/assignments
ls /storm/fubariteblue-ni/workerbeats
```

- Supervisors
- Storms
- Assignments
  - Each topology has one assignment in Zookeeper
    - Contains information we need when restart the topology
    - Supervisor_id => hostname map
    - Executor => node + port map
    - Executor => start-time-secs map
- Workerbeats

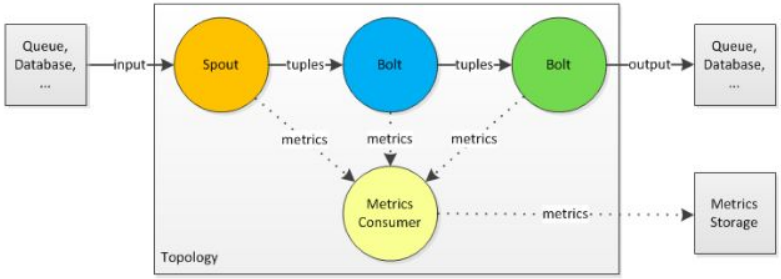
Fault Tolerance



- Workers heartbeat back to supervisors and nimbus via zookeeper
- If workers does (no heartbeat), supervisor will restart it
- If a worker dies repeatedly, nimbus will reassign the work to other nodes in the cluster
- If supervisor node dies, nimbus will ressign work to other nodes
- If nimbus dies, topologies continue to function, but won't be able to perform assignments



Metrics Consumer



- Metrics Consumer Bolt
  - Can listen and handle topology metrics via registering metrics consumer
    - `conf.registerMetricsConsumer(org.apache.storm.metric.LoggingMetricsConsumer.class, 1);`
  - Delegates to MetricsConsumer (customizable) to handle data points via Queue
  - Runs as separate bolt in a separate worker slot
    - https://github.com/apache/storm/blob/master/storm-core/src/jvm/org/apache/storm/metric/MetricsConsumerBolt.java

- Components (Spout/Bolt) can register metrics using IMetric with TaskContext
  - https://github.com/apache/storm/blob/master/storm-core/src/jvm/org/apache/storm/metric/api/IMetric.java
  - Provides various types of counters (CountMetric, CombinedMetric, ReducedMetric etc)

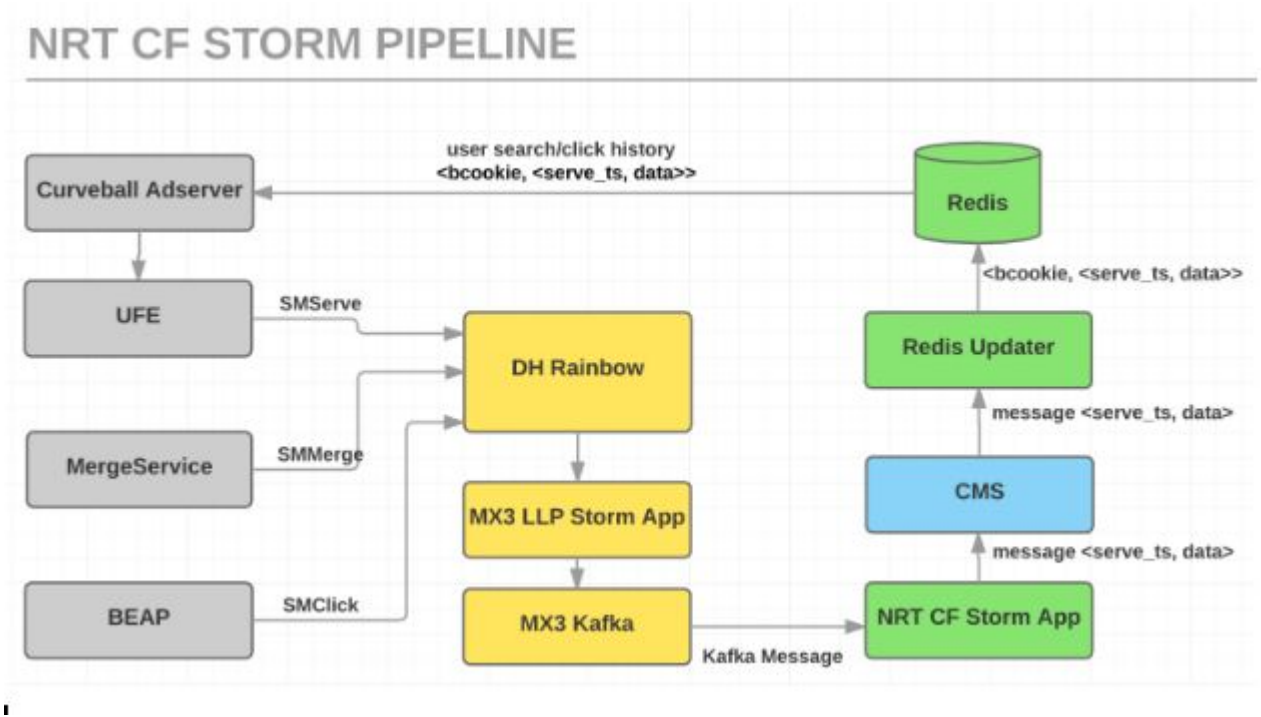
```
public interface IMetric {
    public Object getValueAndReset();
}
```

MetricsConsumerBolt

- Storm appends MetricsConsumerBolt to topology for each registered metrics consumer
- MetricsConsumerBolt subscribes to receive metrics from all tasks
- Storm provides built-in metrics consumers - LoggingMetricsConsumer or implement one
  - Components simply register metric (IMetric) with TopologyContext to publish metric
    - `private transient CountMetric countMetric;`
    - `context.registerMetric("execute_count", countMetric, 60);`
- Built-in Metrics
  - Provides built-in metrics for spouts/bolts (processed, emitted, failed, acked counters etc)
  - Executor (running component tasks) collects metrics from tasks and pushes to MetricsConsumerBolt
- Check Storm UI (Example metric)
  - `2016-07-11 12:02:33.430 b.s.m.LoggingMetricsConsumer __metricsbacktype.storm.metric.LoggingMetricsConsumer_[1 1] [INFO] 1468238553 stbl919n34.blue.ygrid.yahoo.com:6720 5:event_join __sendqueue {arrival_rate_secs=95.8979434730012, overflow=0, read_pos=123153420, write_pos=123153420, sojourn_time_ms=0.0, capacity=1024, population=0}`

NRT Click Feedback - Pipeline

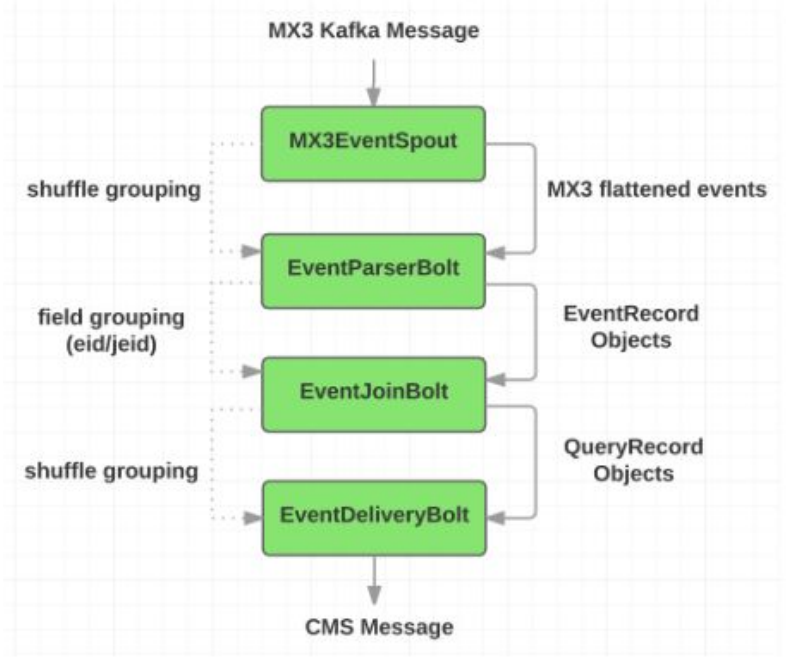
https://docs.google.com/document/d/1j-KKO2RJcas_3hcHqRW6JlSyfkf-8jln467AMEzyzGU/edit#heading=h.ll0b856oiy5s



NRT Storm Topology

https://git.corp.yahoo.com/Curveball/curveball_nrt_click_feedback

https://git.corp.yahoo.com/Curveball/curveball_nrt_click_feedback/blob/master/storm-app/src/main/java/com/yahoo/curveball/nrtcf/stormapp/topology/NrtcfTopology.java



Storm UI - Cluster Summary

http://fubariteblue-ni.blue.ygrid.yahoo.com:9999/index.html

User: srinathm

Storm UI

Cluster Summary

Version	Nimbus uptime	Supervisors	Used slots	Free slots	Total slots	Executors	Tasks
0.10.1.y.138	29d 9h 49m 26s	354	5085	3445	8530	15613	15613

Cluster Resources

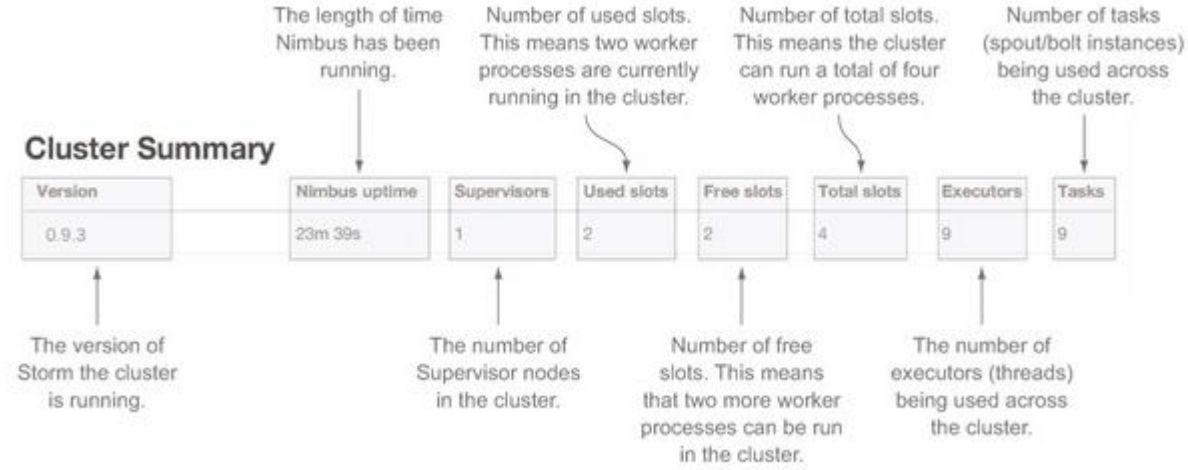
Total Memory (MB)	Total Available Memory (MB)	Memory Utilization (%)	Total CPU (%)	Available CPU (%)	CPU Utilization (%)
31836600	5977592	81.2	813600	312520	61.6

Topology Summary

Show 20 entries

Name	Owner	Status	Uptime	Num workers	Num executors	Num tasks	Assigned Mem (MB)	Assigned CPU (%)
nrtcf	cb_sclkb	ACTIVE	4d 4h 56m 34s	6	6	6	61824	600

Cluster Summary - Information



Supervisor Summary

Show All entries Search:

Id	Host	Uptime	Slots	Used slots	Total Mem (MB)	Used Mem (MB)	Total CPU (%)	Used CPU (%)
9205bba0-fa79-469b-af49-3988c153c49f-10.211.248.158	gsbl866n02.blue.ygrid.yahoo.com	4d 17h 49m 37s	24	4	80800	46080	2300	400
ac320001-56e6-4e5f-9996-c6a7be2ea490-10.211.248.174	gsbl866n03.blue.ygrid.yahoo.com	8d 17h 33m 53s	24	4	90000	65792	2300	400
6fa85451-77e7-45b1-a60b-9a308af597c8-10.211.248.166	gsbl866n04.blue.ygrid.yahoo.com	8d 17h 33m 54s	24	7	90000	87552	2300	760
6d755181-3c0d-4b48-9e9a-3d5d6d8c7d2d-10.211.248.163	gsbl866n05.blue.ygrid.yahoo.com	8d 17h 34m 2s	24	7	90000	87552	2300	860
0b7c94d0-1202-4591-b49c-5ca22e50946c-10.211.248.156	gsbl866n06.blue.ygrid.yahoo.com	8d 17h 33m 42s	24	12	90000	49920	2300	1200

NRT Storm Topology (Spouts/Bolts)

Spouts (All time)

Search:

Id	Executors	Tasks	Emitted	Transferred	Complete latency (ms)	Acked	Failed	Error Host	Error Port	Last error
event	1	1	19545400	19545400	0.000	19545440	0			

Showing 1 to 1 of 1 entries

Bolts (All time)

Search:

Id	Executors	Tasks	Emitted	Transferred	Capacity (last 10m)	Execute latency (ms)	Executed	Process latency (ms)	Acked	Failed	Error Host	Error Port	Last error
event_delivery	1	1	0	0	0.090	0.273	107780100	0.000	0	0			
event_join	2	2	107779900	107779900	0.039	0.017	1157390940	0.000	0	0			
event_parser	1	1	1157393960	1157393960	0.032	0.488	19545440	0.000	0	0			

Showing 1 to 3 of 3 entries

Worker Resources

Search: Toggle Components

Supervisor Id	Host	Port	Uptime	Num executors	Assigned Mem (MB)	Assigned CPU (%)	Components
fc3e88a0-82f2-4eea-9100-cbcb59a62c7b-10.215.72.171	stbl919n34.blue.ygrid.yahoo.com	6720	4d 5h 9m 39s	1	10304	100	1 components
fc3e88a0-82f2-4eea-9100-cbcb59a62c7b-10.215.72.171	stbl919n34.blue.ygrid.yahoo.com	6700	4d 5h 9m 41s	0	10304	100	N/A
fc3e88a0-82f2-4eea-9100-cbcb59a62c7b-10.215.72.171	stbl919n34.blue.ygrid.yahoo.com	6715	4d 5h 9m 41s	1	10304	100	1 components
fc3e88a0-82f2-4eea-9100-cbcb59a62c7b-10.215.72.171	stbl919n34.blue.ygrid.yahoo.com	6705	4d 5h 9m 41s	1	10304	100	1 components
fc3e88a0-82f2-4eea-9100-cbcb59a62c7b-10.215.72.171	stbl919n34.blue.ygrid.yahoo.com	6710	4d 5h 9m 40s	1	10304	100	1 components
fc3e88a0-82f2-4eea-9100-cbcb59a62c7b-10.215.72.171	stbl919n34.blue.ygrid.yahoo.com	6722	4d 5h 9m 42s	1	10304	100	1 components

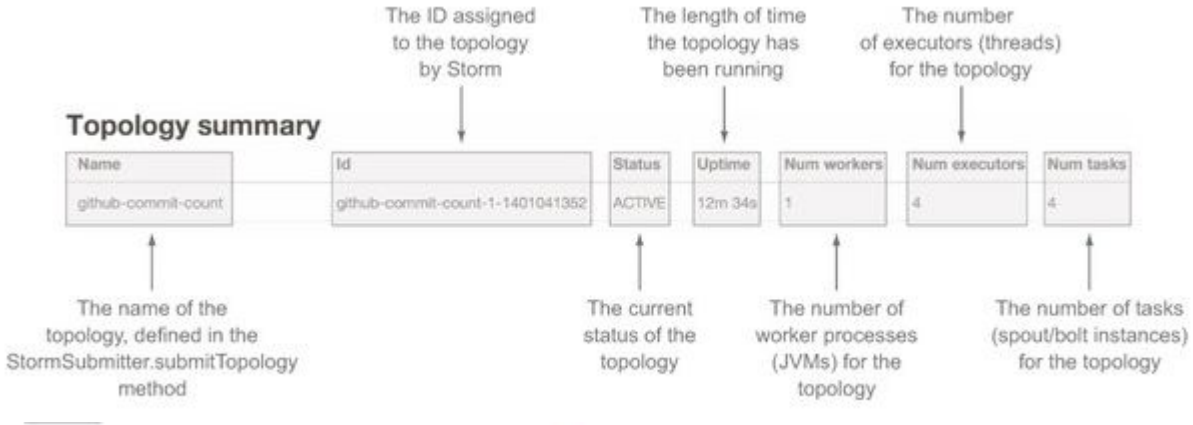
nrtcf - Worker Resources

Worker Resources

Search: Toggle Components

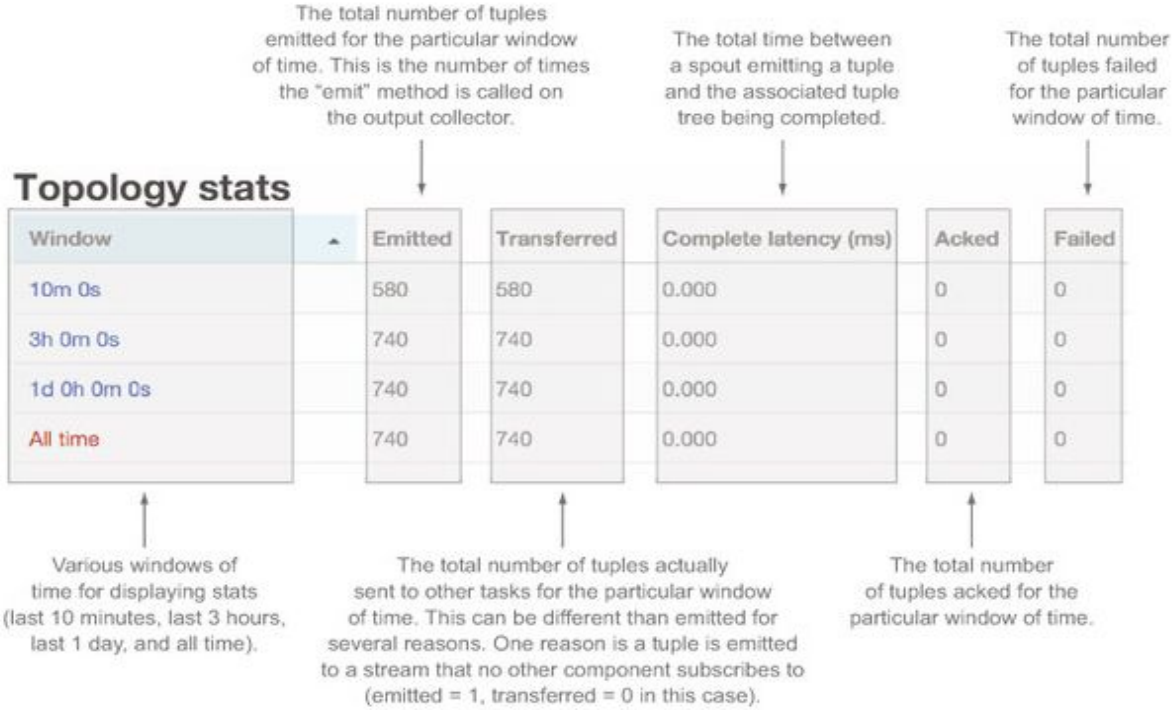
Supervisor Id	Host	Port	Uptime	Num executors	Assigned Mem (MB)	Assigned CPU (%)
fc3e88a0-82f2-4eea-9100-cbcb59a62c7b-10.215.72.171	stbl919n34.blue.ygrid.yahoo.com	6720	4d 5h 18m 16s	1	10304	100
Worker components: event_join 1						
fc3e88a0-82f2-4eea-9100-cbcb59a62c7b-10.215.72.171	stbl919n34.blue.ygrid.yahoo.com	6700	4d 5h 18m 14s	0	10304	100
Worker components: N/A						
fc3e88a0-82f2-4eea-9100-cbcb59a62c7b-10.215.72.171	stbl919n34.blue.ygrid.yahoo.com	6715	4d 5h 18m 14s	1	10304	100
Worker components: event 1						
fc3e88a0-82f2-4eea-9100-cbcb59a62c7b-10.215.72.171	stbl919n34.blue.ygrid.yahoo.com	6705	4d 5h 18m 14s	1	10304	100
Worker components: event_parser 1						
fc3e88a0-82f2-4eea-9100-cbcb59a62c7b-10.215.72.171	stbl919n34.blue.ygrid.yahoo.com	6710	4d 5h 18m 16s	1	10304	100
Worker components: event_delivery 1						
fc3e88a0-82f2-4eea-9100-cbcb59a62c7b-10.215.72.171	stbl919n34.blue.ygrid.yahoo.com	6722	4d 5h 18m 15s	1	10304	100
Worker components: event_join 1						

nrtcf - Topology Summary



Name	Owner	Status	Uptime	Num workers	Num executors	Num tasks	Assigned Mem (MB)	Assigned CPU (%)
nrtcf	cb_sclkb	ACTIVE	4d 4h 56m 34s	6	6	6	61824	600

Topology Stats



Topology stats

Window	Emitted	Transferred	Complete latency (ms)	Acked
10m 0s	2390853	2390853	0.000	36646
3h 0m 0s	52068885	52068885	0.000	793894
1d 0h 0m 0s	331566471	331566471	0.000	5166901
All time	1291633820	1291633820	0.000	19651480

“event” spout stats in nrtcf topology

Component summary

Id	Topology	Executors	Tasks
event	nrtcf	1	1

Spout stats

Window	Emitted	Transferred	Complete latency (ms)	Acked	Failed
10m 0s	35692	35692	0.000	35659	0
3h 0m 0s	786153	786153	0.000	786024	0
1d 0h 0m 0s	5172467	5172467	0.000	5172614	0
All time	19678540	19678540	0.000	19678520	0

Output stats (All time)

Search:

Stream	Emitted	Transferred	Complete latency (ms)	Acked	Failed
default	19678540	19678540	0.000	19678520	0

“event_parser” bolt stats in nrtcf Topology

- Pay attention to executed vs emitted

Component summary

Id	Topology	Executors	Tasks
event_parser	nrtcf	1	1

Bolt stats

Search:

Window	Emitted	Transferred	Execute latency (ms)	Executed	Process latency (ms)	Acked
10m 0s	2076029	2076029	0.474	35256	0.000	0
3h 0m 0s	45874605	45874605	0.490	779043	0.000	0
1d 0h 0m 0s	298904976	298904976	0.495	5177015	0.000	0
All time	1166478960	1166478960	0.488	19700320	0.000	0

“event_join” bolt stats in nrtcf topology

Component summary

Id	Topology	Executors	Tasks
event_join	nrtcf	2	2

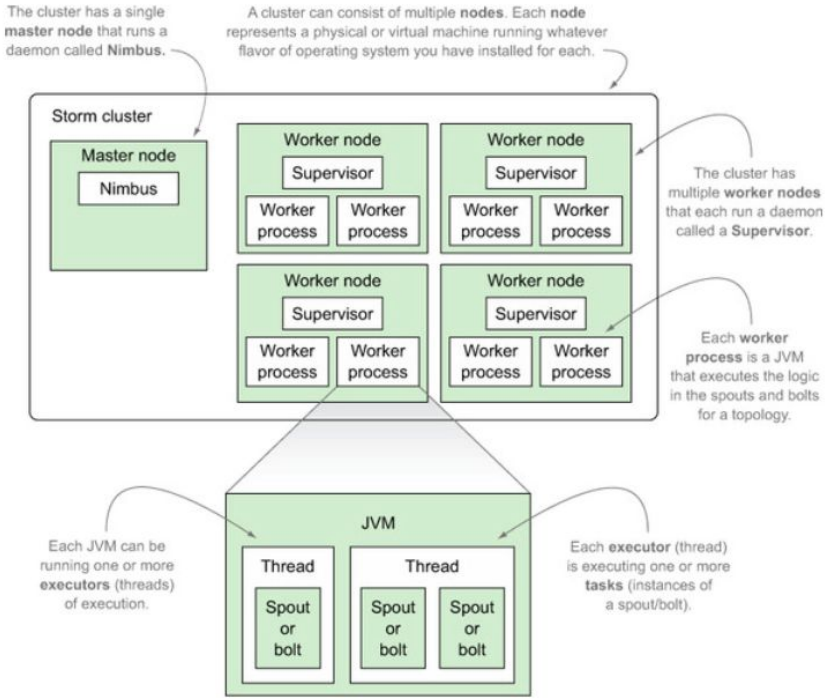
Bolt stats

Search:

Window	Emitted	Transferred	Execute latency (ms)	Executed	Process latency (ms)	Acked
10m 0s	197662	197662	0.015	2046620	0.000	0
3h 0m 0s	4393425	4393425	0.017	45623804	0.000	0
1d 0h 0m 0s	28131368	28131368	0.017	299014161	0.000	0
All time	108721220	108721220	0.017	1167158540	0.000	0

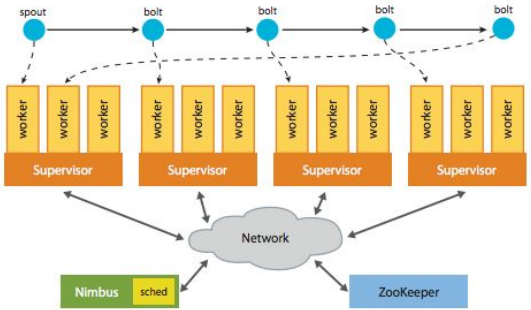
- Pay attention to Executed vs Emitted

Cluster Summary



- Configure the number of worker processes being run on the worker node
- supervisor.slots.ports property controls the number of worker processes on a given worker node
- Defines the ports that each worker process will use to listen for messages

Topology - Parallelism

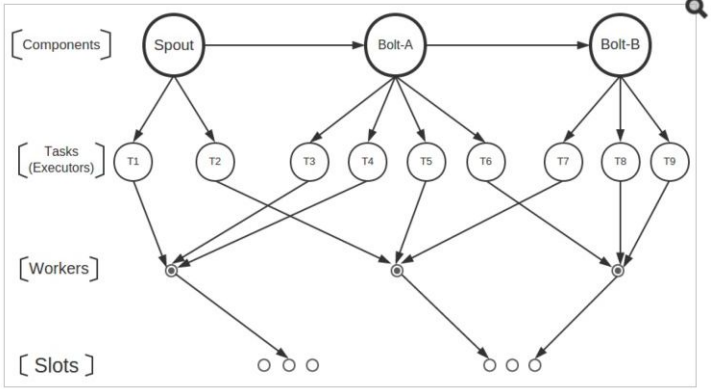


- Worker Processes
  - How many worker processes to create for the topology across machines in the cluster
  - conf.setNumWorkers(x) - Topology level configuration
- Executors
  - How many executors to spawn per component
- Tasks
  - How many tasks to create per component (2 executors and 4 tasks)
  - ```
    topologyBuilder.setBolt("green-bolt", new GreenBolt(), 2)
                   .setNumTasks(4)
                   .shuffleGrouping("blue-spout");
    ```
- Rebalancing topology

```
# Reconfigure the topology "mytopology" to use 5 worker processes,
# the spout "blue-spout" to use 3 executors and
# the bolt "yellow-bolt" to use 10 executors.

$ storm rebalance mytopology -n 5 -e blue-spout=3 -e yellow-bolt=10
```
```

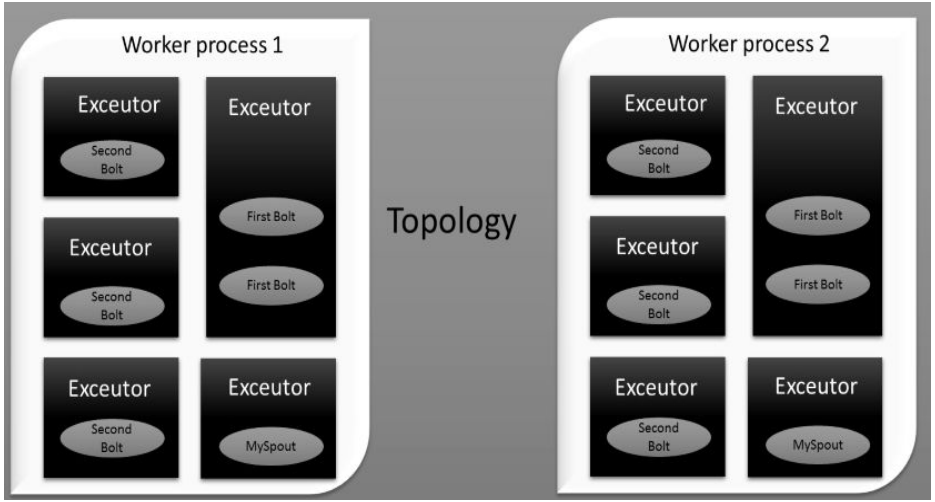
Scheduling in Storm



- A component have one or more tasks
- Tasks are assigned to one or more executors
- Executors are assigned to one or more workers
- Each worker is then assigned into worker slot (Scheduler Interface)
- Storm Scheduler
  - Decides which tasks can be assigned into same worker (based on topology)
  - Task scheduling abides by both component order and grouping specifications
- Scheduling Algorithm
  - Which worker is assigned to which slot based on scheduling algorithm

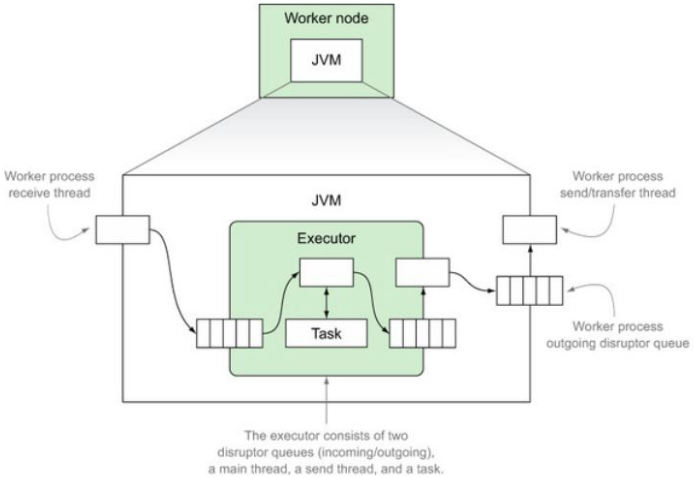
Example Topology

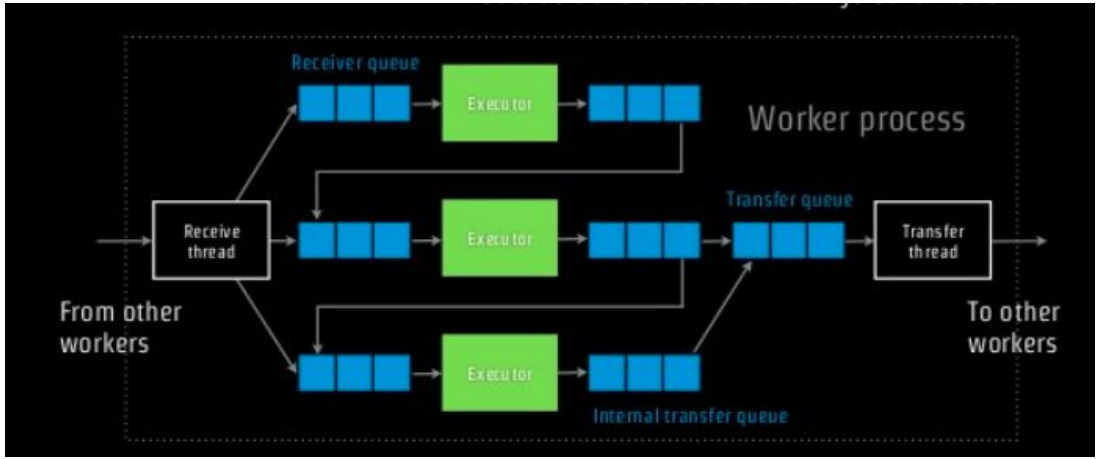
```
Config topologyConf = new Config();
topologyConf.setNumWorkers(2);
topologyBuilder.setSpout("my-spout", new MySpout(), 2);
topologyBuilder.setBolt("first-bolt", new FirstBolt(),2)
.setNumTasks(4)
.shuffleGrouping("my-spout");
topologyBuilder.setBolt("second-bolt", new YellowSecondBolt(), 6)
               .shuffleGrouping("first-bolt");
```



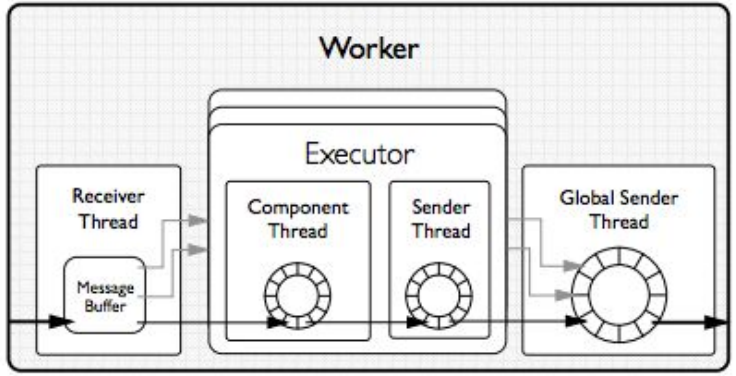
- Total combined parallelism (number of executors) for all components = 2 (spout parallelism) + 2 (first bolt parallelism) + 6 (second bolt parallelism) = 10
- Number of workers allocated = 2
- Number of executors to be spawned on each worker = 10/2 = 5
- Number of tasks for spout = 1 (default)
- Number of tasks for first bolt = 4
- Number of tasks for second bolt = 6

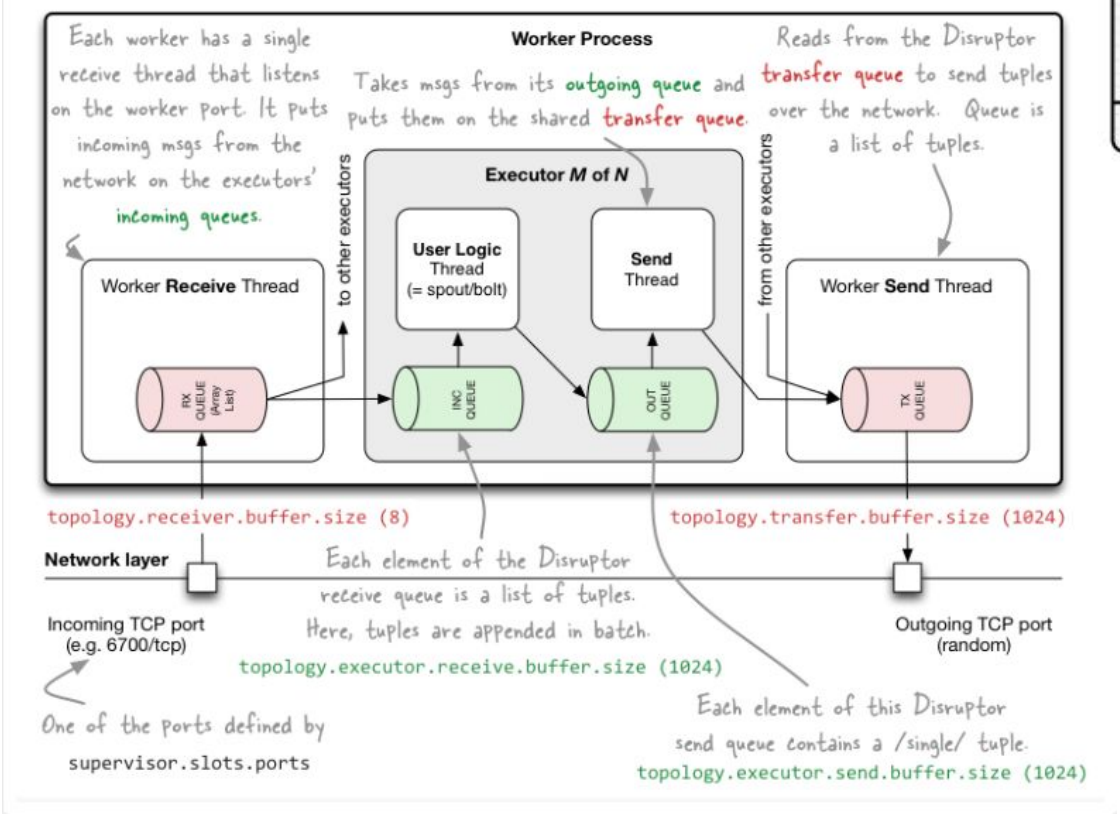
Worker Node - Topology Execution



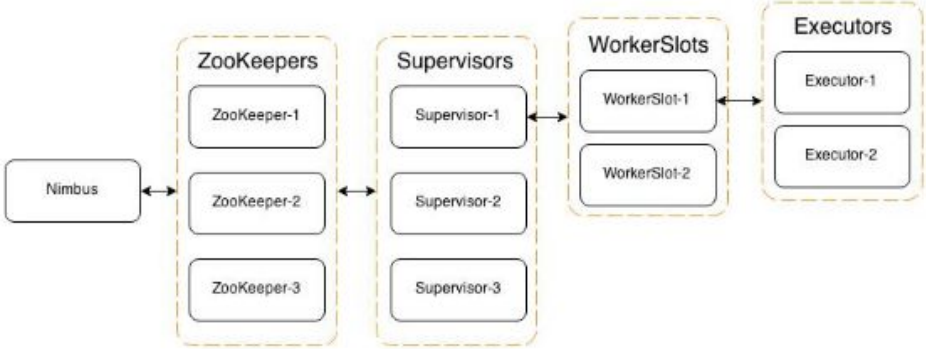


Routing in Worker Node



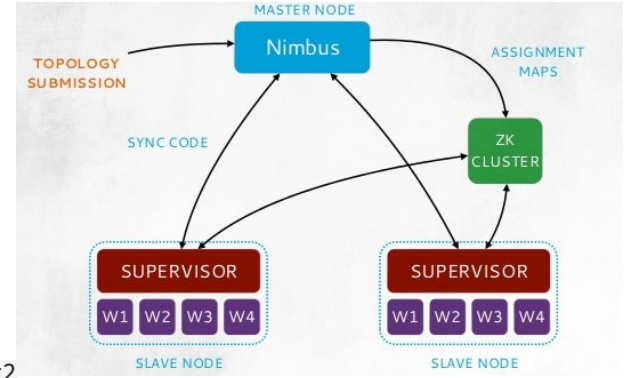


Scheduling



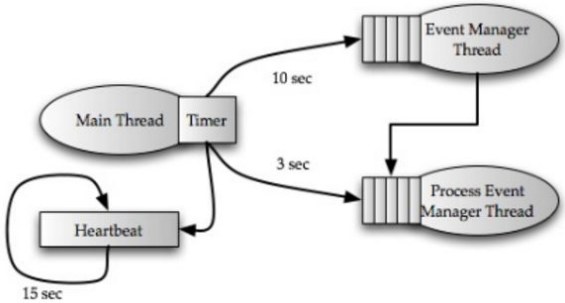
- Supervisor’s computing resources are partitioned into “WorkerSlots”
- In each slot, Storm can spawn multiple threads - Executors
  - fulfill tasks assigned by Nimbus
  - Nimbus uses a scheduler to assign tasks to supervisors
- Nimbus
  - Responsible for distributing and coordinating the execution of topology
  - Cluster state is maintained in zookeeper

Nimbus / Supervisors

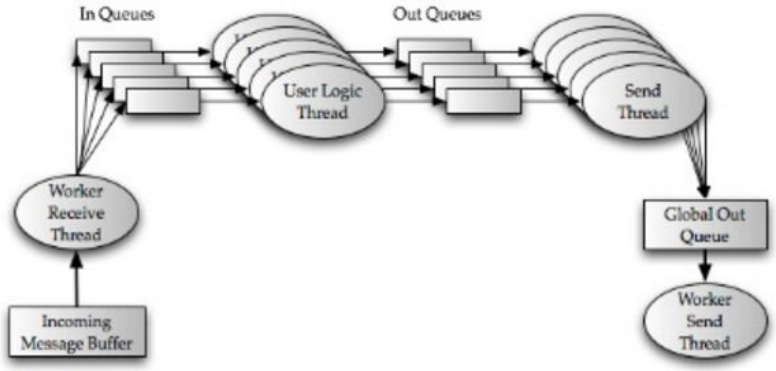


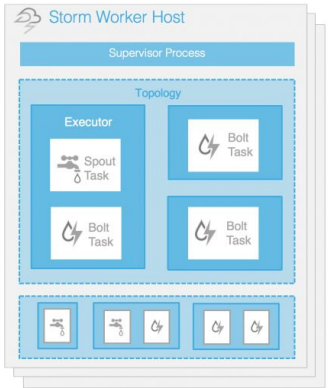
- Nimbus plays a similar role “JobTracker” in Hadoop
  - Touchpoint between the user and storm
    - `storm jar all-my-code.jar org.apache.storm.MyTopology arg1 arg2`
  - Submits topology to Nimbus (User code as JAR file)
  - Nimbus maintains the state of topology in zookeeper (active worker nodes)
  - Supervisors contact Nimbus with a periodic heartbeat protocol
    - Advertising topologies currently running (any vacancies etc)
- Supervisors
  - Runs on each storm node, receives assignments from Nimbus
  - Spawns workers based on assignments
    - Heartbeat Event - Reports to Nimbus (every 15 seconds)
    - Synchronize Supervisor Event - Manages assignments
      - Downloads Topology jar file and schedules synchronize process event
    - Synchronize process event
      - Managing worker process that run a fragment of topology

Supervisor Architecture



Message flow inside worker





- Runs on each storm node, receives assignments from Nimbus
- Spawns workers based on assignments
  - Heartbeat Event - Reports to Nimbus (every 15 seconds)
  - Synchronize Supervisor Event - Manages assignments
    - Downloads Topology jar file and schedules synchronize process event
  - Synchronize process event
    - Managing worker process that run a fragment of topology

Guaranteed Processing

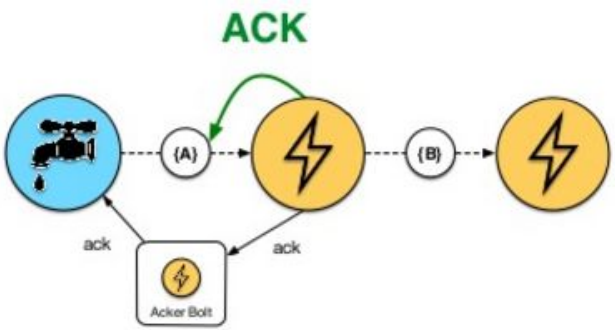
- Provides API to guarantee that a tuple emitted by spout is fully processed by topology
- A tuple coming off a spot can trigger thousands of tuples to be created based on it
  - Let’s take a look at WordCount Storm topology example
    - Spout generates sentences (tuples)
    - Bolt generates a set of words for each sentence (child tuples)
- A tuple is fully processed if all its child tuples have been correctly processed
- With guaranteed processing, each bolt in the tree can either ack or fail a tuple
  - If all bolts in the tree ack the tuple and child tuples, the msg processing is complete
  - If any bolt explicitly fail a tuple (or timeout), the processing is failed
- Form spout side
  - Need to keep track of the tuple emitted and be prepared to handle failures
- On bolt side
  - Anchor any emitted tuple to the original one and ack the original tuple (or fail)

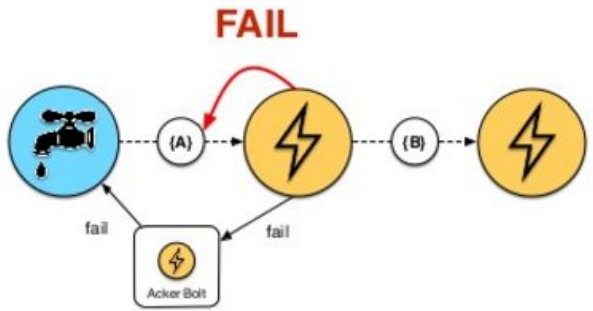
XOR Trick (refresh memory)

- Long a, b, c = Random.nextLong()
  - a ^ a == 0
  - a ^ a ^ b != 0
  - a ^ a ^ b ^ b == 0
  - a ^ (a ^ b) ^ c ^ (b ^ c) == 0
- Acks can arrive asynchronously in any order

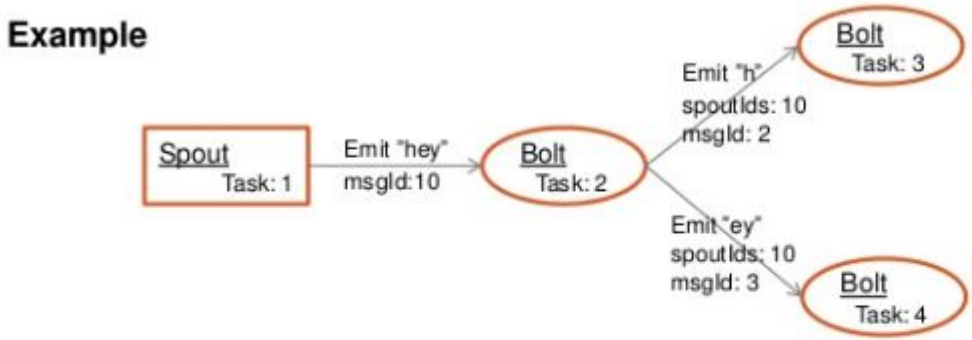
Reliable Processing in STORM - Implementation

- Storm’s acker tracks completion of each tuple tree with checksum hash
  - When a tuple is created, it is given random 64bit id
    - Acker (system level bolt) stores a map spout tuple id -> [spout id, ack value]
    - Mod hashing is used to map a spout tuple id to an acker task
  - Ack value is 64 bit = XOR of all tuple ids, represents the state of tuple tree
    - Each time a tuple is sent, its value is XORed into the ack value, and each acked its value is XORed in again
    - If all tuples have been successfully acked, the ack value (checksum) will be zero





Simple Example - ACK

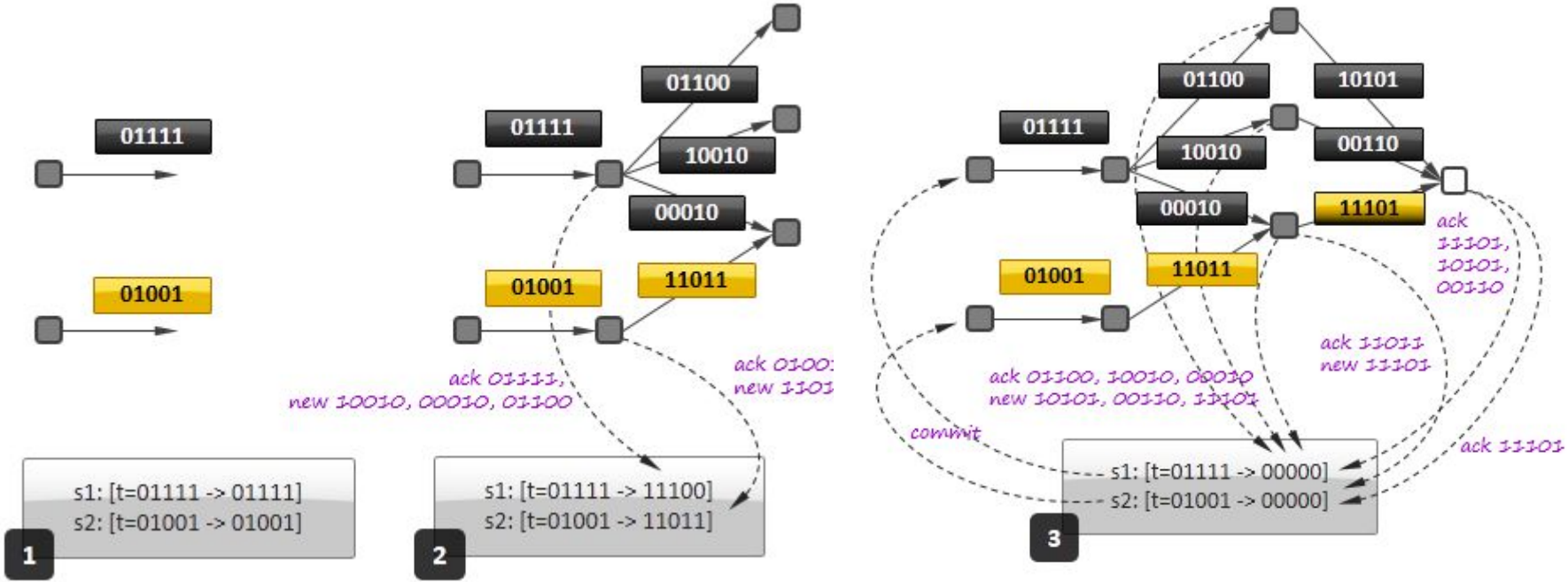


Shows what happens in acker task, for one spout tuple. Format is: {spoutMsgId, {spoutTaskId, "ack val"}}

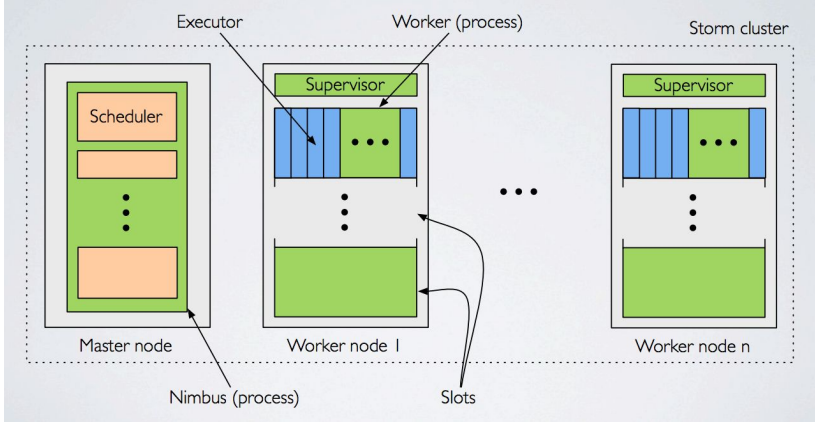
1. After emit "hey": {10, {1, 0000 XOR 1010 = 1010}
2. After emit "h": {10, {1, 1010 XOR 0010 = 1000}
3. After emit "ey": {10, {1, 1000 XOR 0011 = 1011}
4. After ack "hey": {10, {1, 1011 XOR 1010 = 0001}
5. After ack "h": {10, {1, 0001 XOR 0010 = 0011}
6. After ack "ey": {10, {1, 0011 XOR 0011 = 0000}
7. Since "ack val" is 0, spout tuple with id 10, must be fully processed. Call ack on spout (task 1)

Ack Processing in STORM

https://github.com/apache/storm/blob/master/storm-core/src/jvm/org/apache/storm/daemon/Acker.java



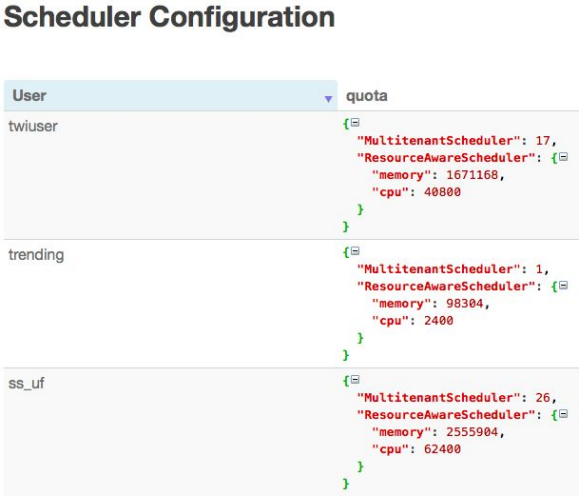
Scheduler



- A topology is run by submitting to Nimbus
- Nimbus allocates the execution of components (Spouts and Bolts) to the worker nodes using scheduler
  - Each component has multiple instances (parallelism)
  - Each instance is mapped to an executor
- A worker is instantiated whenever the hosting node must run executors for the submitted topology

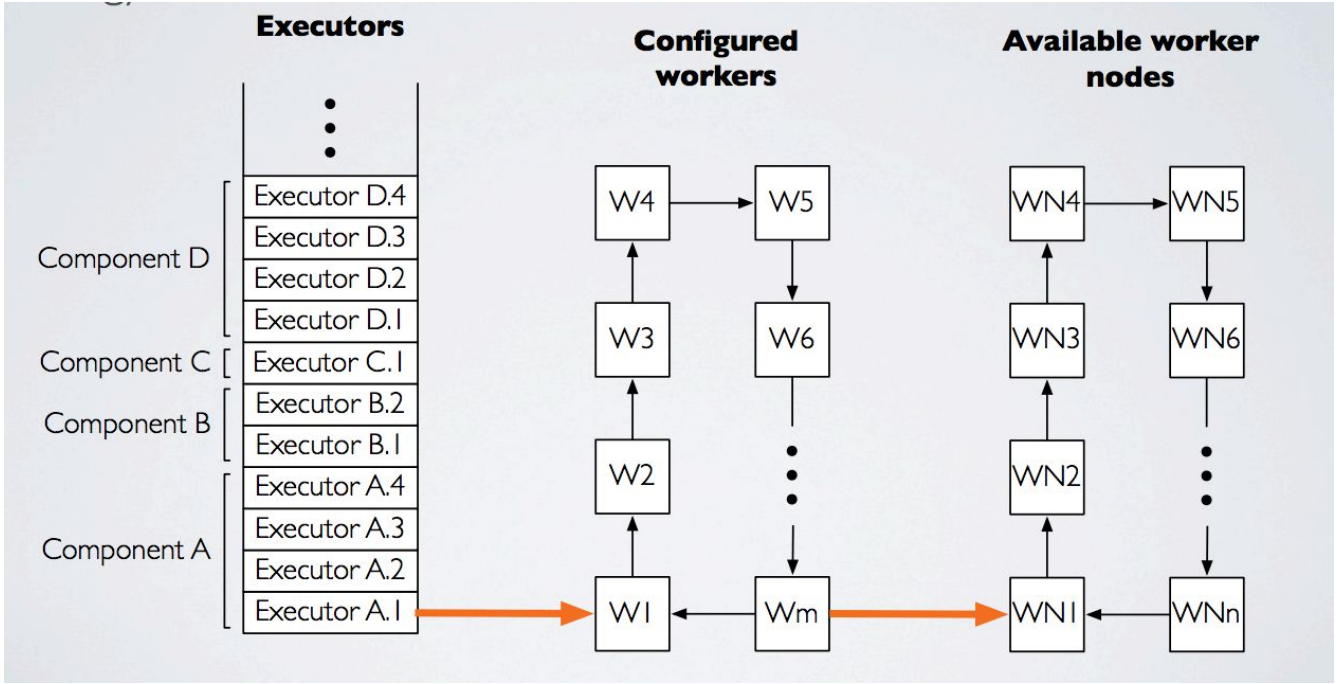
Pluggable Schedulers

- Storm supports multiple types of schedulers
  - Even Scheduler
    - Evenly assigns topologies to worker nodes
      - Round Robin Strategy
  - Isolation Scheduler
    - Assign predefined topologies to a dedicated subset of worker nodes
  - Multi-tenant Scheduler
    - Provides admin resource allotments per user instead of per topology
    - Users decide how to divide up their resources per topology
  - Resource Aware Scheduler (RAS)
    - Matching task resource requirements to resources available on Node
  - Multi-tenant Resource Aware Bride Scheduler
    - Combination of multi-tenant scheduler and RAS



storm.scheduler	"backtype.storm.scheduler.bridge.MultitenantResourceAwareBridgeScheduler"

Even scheduler - Round Robin Strategy

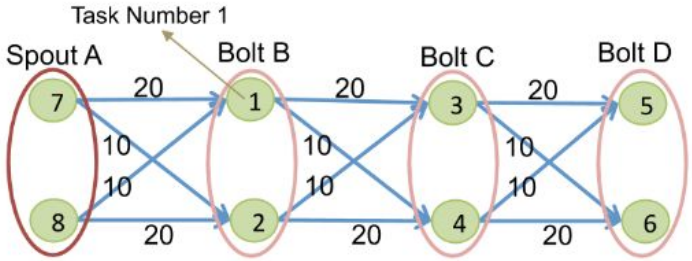


Scheduler

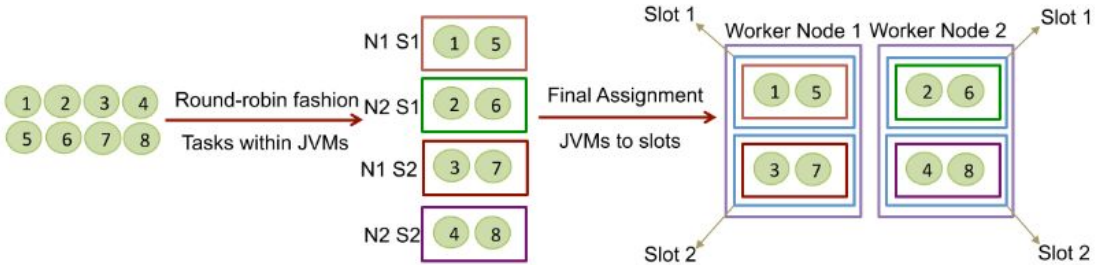
- Implementing custom scheduler in Storm (IScheduler interface)
  - public void schedule(Topologies topologies, Cluster cluster)
- Topologies - Contains collection of TopologyDetails instances (getTopologies)
- Cluster - Collection of SupervisorDetails instances (getSupervisiors)
- Check whether topology requires scheduling?
  - cluster.needsScheduling(TopologyDetails topology)
  - Spouts/Bolts in a topology are encapsualted in StormTopology (getTopology)
  - Bolt (get_bolts on StormTopology), SpoutSpec (get_spouts on StormTopology)
  - Assign a component to available WorkerSlot on SuperVisorDetails (getAvailableSlots)
  - Get executors (to be scheduled) for each component - Map<name, ExecutorDetails>
    - getNeedsSchedulingComponentToExecutor(TopologyDetails topology)
    - Get available slots on supervisor (getAvailableSlots(SupervisiorDetails sd))
    - Finally use assign WorkerSlot to Executors
      - cluster.assign(workerslot, topologyId, executors)

Task Assignment - Default Scheduler

https://github.com/apache/storm/blob/master/storm-core/src/jvm/org/apache/storm/scheduler/DefaultScheduler.java



(a) An example of Storm topology



(b) Task assignment in two phases

MX3 LLP SearchTrafficSink Bolt

- SearchTrafficSink Bolt
  - Publishes SM_SERVE, SM_MERGE and SM_CLICKS to SearchTraffic Kafka Topic
  - Applied ad_listing_filter to remove truncated and deduped events
  - SearchTrafficRecord captures SM Event (Serve/Merge/Click)
    - Creates a batch of SearchTrafficRecords (`SerializedSearchTrafficRecordList`)
    - Publishes one batch at a time to Kafka topic
    - https://git.corp.yahoo.com/MX3/llp_nimbus/blob/master/src/main/java/yahoo/curveball/mx3llp/operators/SearchTrafficSinkBolt.java
    -

NRTCF Pipeline - Kafka Spout

- Consumes SearlizedSearchTrafficRecords from SearchTraffic Kafka Topic
  - https://git.corp.yahoo.com/Curveball/curveball_nrt_click_feedback/blob/master/storm-app/src/main/java/com/yahoo/curveball/nrtcf/stormapp/topology/NrtcfTopology.java

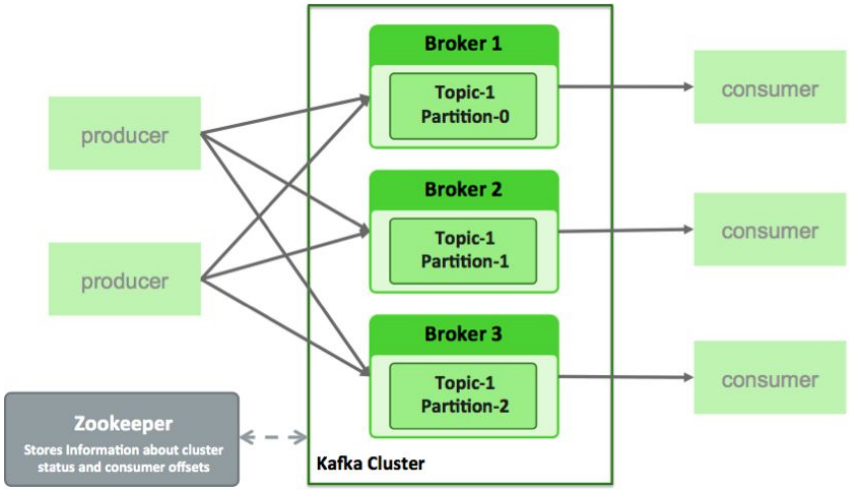
```
String kafkaZk = configUtils.get(Constants.KafkaSpout.MX3LLP_CONSUMER_ZOOKEEPERS);
String kafkaGroupId = configUtils.get(Constants.KafkaSpout.MX3LLP_CONSUMER_GROUP_ID);
String kafkaTopic = configUtils.get(Constants.KafkaSpout.MX3LLP_CONSUMER_TOPIC);
String kafkaZkRoot = configUtils.get(Constants.KafkaSpout.MX3LLP_CONSUMER_ZOOKEEPER_ROOT, "");
SpoutConfig spoutConf = new SpoutConfig(new ZkHosts(kafkaZk), kafkaTopic, kafkaZkRoot, kafkaGroupId);

spoutConf.ignoreZkOffsets = true;
spoutConf.startOffsetTime = kafka.api.OffsetRequest.LatestTime();
KafkaSpout kafkaSpout = new KafkaSpout(spoutConf);

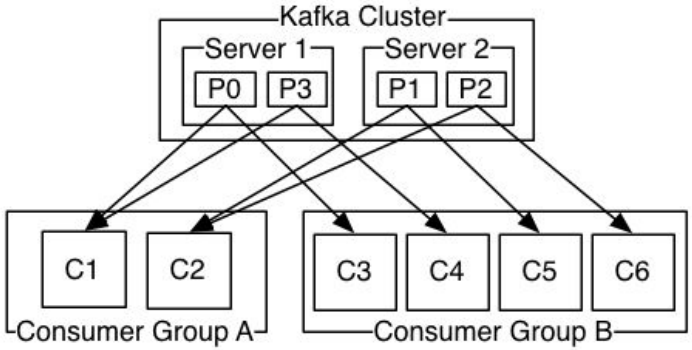
int parallelism = configUtils.get(Constants.KafkaSpout.PARALLELISM, Constants.Topology.DEFAULT_PARALLELISM);
int numOfCPUs = configUtils.get(Constants.KafkaSpout.CPU, Constants.KafkaSpout.DEFAULT_CPU);
int memory = configUtils.get(Constants.KafkaSpout.MEMORY, Constants.KafkaSpout.DEFAULT_MEMORY);

SpoutDeclarer spoutDeclarer = topologyBuilder.setSpout(Constants.KafkaSpout.ID, kafkaSpout, parallelism);
spoutDeclarer.setCPULoad(numOfCPUs);
spoutDeclarer.setMemoryLoad(memory);
```

Quick Tour - Kafka Cluster



- Producers write data to brokers
- Consumers read data from brokers
- Brokers persist data to disk for sometime
- Data
  - Data is stored in Topics
  - Topics are split into partitions which are replicated
  - Retains all published messages for a configurable amount of time
- Example
  - Two Brokers cluster
  - One topic - 4 partitions (2 on each broker)
  - Two consumer groups
    - One consumer group - 2 consumers
    - Another group - 4 consumers



Explore MX3 Kafka Cluster

- GQ1 Zookeepers
  - kafka1.dp.cb.gq1.yahoo.com, kafka2.dp.cb.gq1.yahoo.com
- Connect to one of these zookeepers and explore cluster state
  - -bash-4.1$ zoosh kafka1.dp.cb.gq1.yahoo.com:2181
  - SearchTraffic captures search traffic events
  - 15 partitions (ls brokers/topics/SearchTraffic/partitions)
  - 5 Brokers cluster

```
>>  cat brokers/ids/1
{"jmx_port":-1,"timestamp":"1461857453599","host":"kafka3.dp.cb.gq1.yahoo.com","version":1,"port":9092}
>>  cat brokers/ids/9
{"jmx_port":-1,"timestamp":"1462251663766","host":"kafka6.dp.cb.gq1.yahoo.com","version":1,"port":9092}
>>  cat brokers/ids/8
{"jmx_port":-1,"timestamp":"1465274317001","host":"kafka5.dp.cb.gq1.yahoo.com","version":1,"port":9092}
>>  cat brokers/ids/5
{"jmx_port":-1,"timestamp":"1465402067146","host":"splunkhd2.dp.cb.gq1.yahoo.com","version":1,"port":9092}
>>  cat brokers/ids/4
{"jmx_port":-1,"timestamp":"1462381035959","host":"kafka4.dp.cb.gq1.yahoo.com","version":1,"port":9092}
```

```
-bash-4.1$ zoosh kafka1.dp.cb.gq1.yahoo.com:2181
Using servers [kafka1.dp.cb.gq1.yahoo.com:2181]
Initialized zookeeper handle (0x24518d0)
INFO  zookeeper connected
Got a new id: 15512159b1b867f
>>  ls brokers/topics
    GeminiConvs
    __consumer_offsets
    GeminiClicks
    RawSupply
    SearchTraffic
    EligibleEvent
    RawDXS
    SupplyAgg
    AppInstalls
    Budget
    EligibleEventAI
    DXSPreAgg
```

SearchTraffic Topic Partitions

- Search Traffic topic has 15 partitions
  - Each partition is replicated 3 times
  - At any time one broker acts as leader - all requests being sent to the leader
  - For example, partition 0 is hosted by brokers 1, 4 and 9 (broker ids)
    - Broker 1 is the leader

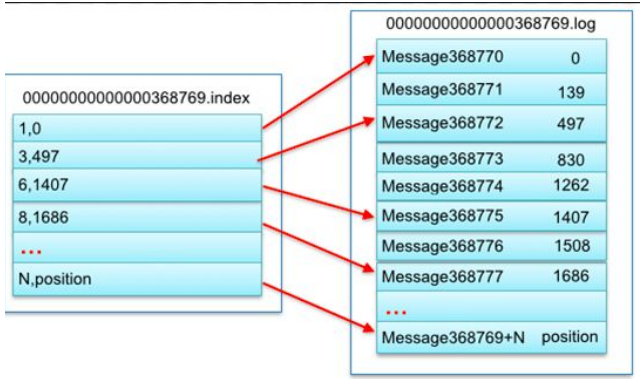
```
>>  cat brokers/topics/SearchTraffic
{"version":1,"partitions":{"12":[8,9,1],"8":[9,8,1],"4":[1,8,9],"11":[5,8,9],"9":[1,9,4],
"13":[9,1,4],"5":[4,1,5],"10":[4,5,8],"6":[5,4,8],"1":[5,1,4],"14":[1,4,5],
"0":[4,9,1],"2":[8,4,5],"7":[8,5,9],"3":[9,5,8]}}

>>  cat brokers/topics/SearchTraffic/partitions/0/state
{"controller_epoch":15,"leader":1,"version":1,"leader_epoch":2,"isr":[1,4,9]}
```

Broker Storage

For example, a partition has below 4 segments.

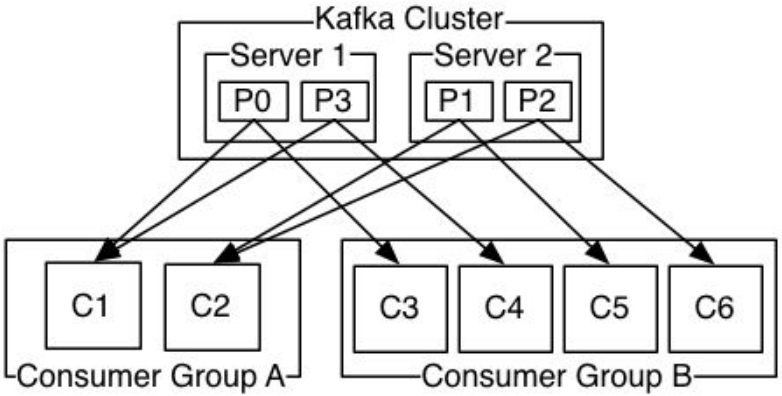
```
00000000000000000000.index
00000000000000000000.log
00000000000000368769.index
00000000000000368769.log
00000000000000737337.index
00000000000000737337.log
00000000000001105814.index
00000000000001105814.log
```



- Topic Storage
  - Each partition of a topic corresponds to a logical log
  - A log is implemented as a set of segment files (1 GB in size)
  - Broker simply appends the message to the segment file
  - Each message is addressed by its logical offset in log file (no message id)
  - Consumer always consumes messages from a particular partition sequentially (Pull request)
  - Broker keeps in-memory sorted list of offsets - segment file location

Kafka Consumer Group

- A group of consumers jointly consume a set of topics
  - Each message is delivered to only one of consumers in a group
  - Consumers with in same group can be in different processes or on different machines
  - Load balancing in consumer group



Kafka Spout

- No ConsumerGroup, but it uses SimpleConsumer to have greater control over partition consumption than ConsumerGroup gives us.
  - Example: Consume only a subset of partitions in a topic in a process
  - Manage transactions to make sure message is delivered only once
- SimpleConsumer
  - Client must keep track of the offsets in the application (where you left off consuming)
  - Client need to know which broker is leader for a given topic and partition
  - Client need to take care of broker leader changes
- Using Simple Consumer
  - Determine the broker for the topic, partition
  - Build request and fetch data from broker hosting partition
  - Identify and recover from leader changes
  - https://cwiki.apache.org/confluence/display/KAFKA/0.8.0+SimpleConsumer+Example

Kafka Spout - Partition Management

- Kafka Spout Parallelism
  - Determines what partitions to read from
  - Deterministic parallelism based on task id
  - Example Kafka Spout Parallelism: 4 Executors and 2 tasks each
    - Total tasks = 4*2 = 8 tasks
    - Let’s say Kafka topic has 16 partitions, then each task instance reads 2 partitions
    - Task (0, 1) reads from partitions (0, 1)
- Kafka Spout internals
  - Identify leader for each partition (from zookeeper) - Broker (host, port)
  - Create Partition Manager for each of the partitions
  - PartitionManager encapsulates SimpleConsumer connecting to Broker
  - nextTuple fetches data from each partition (Round Robin Strategy)
- https://github.com/apache/storm/blob/master/external/storm-kafka/src/jvm/org/apache/storm/kafka/KafkaSpout.java

SimpleConsumer API

```
class kafka.javaapi.consumer.SimpleConsumer {
  /**
   *  Fetch a set of messages from a topic.
   *
   *  @param request specifies the topic name, topic partition, starting byte offset, maximum bytes to be fetche
   *  @return a set of fetched messages
   */
  public FetchResponse fetch(kafka.javaapi.FetchRequest request);

  /**
   *  Fetch metadata for a sequence of topics.
   *
   *  @param request specifies the versionId, clientId, sequence of topics.
   *  @return metadata for each topic in the request.
   */
  public kafka.javaapi.TopicMetadataResponse send(kafka.javaapi.TopicMetadataRequest request);

  /**
   *  Get a list of valid offsets (up to maxSize) before the given time.
   *
   *  @param request a [[kafka.javaapi.OffsetRequest]] object.
   *  @return a [[kafka.javaapi.OffsetResponse]] object.
   */
  public kafak.javaapi.OffsetResponse getOffsetsBefore(OffsetRequest request);

  /**
   * Close the SimpleConsumer.
   */
  public void close();
}
```

SearchTraffic Kafka Topic Spout

- SearchTraffic Kafka Topic
  - Has 15 partitions hosted on 5 brokers (3 replicas each)
- SearchTraffic Kafka Topic Spout
  - Has only one executor (parallelism = 1), so all partitions assigned to this task
  - nextTuple reads from all 15 partitions in a loop
  - Msg offset is getting persisted for each partition in zookeeper
    - http://fubariteblue-ni.blue.ygrid.yahoo.com:9999/topology.html?id=nrtcf-316-1467408706

Id	Executors	Tasks	Emitted	Transferred	Complete latency (ms)	Acked
event	1	1	41965100	41965100	0.000	41965120

```
get /nrtcf/partition_1
{"topology":{"id":"028110a5-5386-4512-b9c1-2fec8957815c",
 "name":"nrtcf"},
 "offset":24056292,
 "partition":1,
 "broker":{"host":"kafka3.dp.cb.gq1.yahoo.com",
 "port":9092},"topic":"SearchTraffic"}
```

```
2016-07-10 18:03:24.407 s.k.DynamicBrokersReader event_[2 2] [INFO] Read partition info from zookeeper:
GlobalPartitionInformation{partitionMap={0=kafka3.dp.cb.gq1.yahoo.com:9092, 1=kafka3.dp.cb.gq1.yahoo.com:9092,
2=kafka4.dp.cb.gq1.yahoo.com:9092, 3=kafka6.dp.cb.gq1.yahoo.com:9092, 4=kafka6.dp.cb.gq1.yahoo.com:9092,
5=kafka3.dp.cb.gq1.yahoo.com:9092, 6=kafka4.dp.cb.gq1.yahoo.com:9092, 7=kafka6.dp.cb.gq1.yahoo.com:9092,
8=kafka6.dp.cb.gq1.yahoo.com:9092, 9=kafka3.dp.cb.gq1.yahoo.com:9092, 10=kafka4.dp.cb.gq1.yahoo.com:9092,
11=kafka6.dp.cb.gq1.yahoo.com:9092, 12=kafka6.dp.cb.gq1.yahoo.com:9092,
13=kafka3.dp.cb.gq1.yahoo.com:9092, 14=kafka3.dp.cb.gq1.yahoo.com:9092}}

2016-07-10 18:03:24.407 s.k.KafkaUtils event_[2 2] [INFO] Task [1/1] assigned [
Partition{host=kafka3.dp.cb.gq1.yahoo.com:9092, partition=0},
Partition{host=kafka3.dp.cb.gq1.yahoo.com:9092, partition=1},
Partition{host=kafka4.dp.cb.gq1.yahoo.com:9092, partition=2},
Partition{host=kafka6.dp.cb.gq1.yahoo.com:9092, partition=3},
...
Partition{host=kafka3.dp.cb.gq1.yahoo.com:9092, partition=13},
Partition{host=kafka3.dp.cb.gq1.yahoo.com:9092, partition=14}]
```

Nrtcf Topology

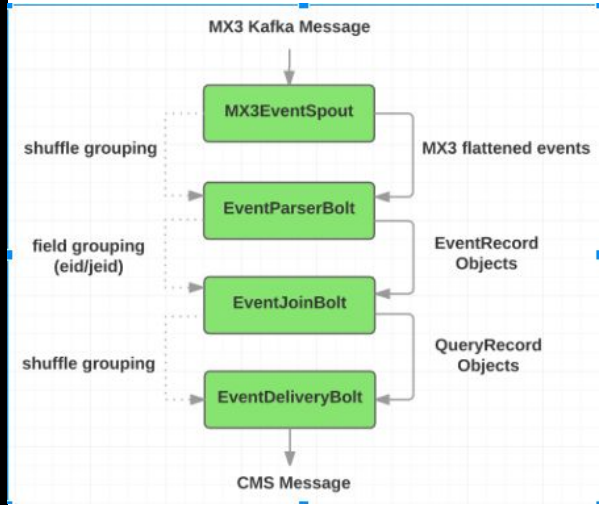
- https://git.corp.yahoo.com/Curveball/curveball_nrt_click_feedback/blob/master/storm-app/src/main/java/com/yahoo/curveball/nrtcf/stormapp/topology/NrtcfTopology.java

```
KafkaSpout kafkaSpout = new KafkaSpout(spoutConf);
topologyBuilder.setSpout(Constants.KafkaSpout.ID, kafkaSpout, 1);

EventParserBolt eventParserBolt = new EventParserBolt(configUtils);
topologyBuilder.setBolt(Constants.EventParserBolt.ID, eventParserBolt, 1)
               .shuffleGrouping(Constants.KafkaSpout.ID);

EventJoinBolt eventJoinBolt = new EventJoinBolt(configUtils);
topologyBuilder.setBolt(Constants.EventJoinBolt.ID, eventJoinBolt, 2)
               .fieldsGrouping(Constants.EventParserBolt.ID,
                 Constants.EventParserBolt.STREAM,
                 new Fields(Constants.EventParserBolt.EVENT_ID));

EventDeliveryBolt eventDeliveryBolt = new EventDeliveryBolt(configUtils);
topologyBuilder.setBolt(Constants.EventDeliveryBolt.ID, eventDeliveryBolt, 1)
               .shuffleGrouping(Constants.EventJoinBolt.ID,
                                Constants.EventJoinBolt.STREAM);
```



Kafka Spout Consumer Info

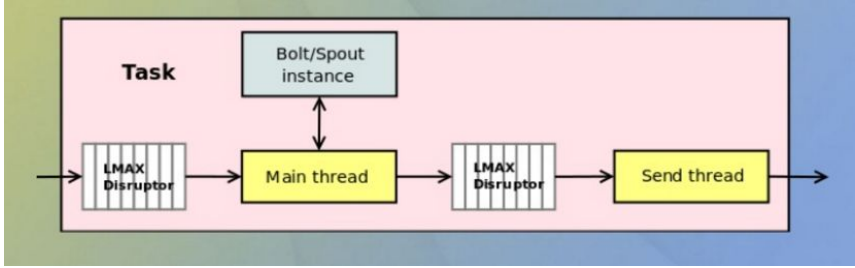
- Connect to storm zookeeper host
  - Ssh to pb host: ssh phazon-gw.blue.ygrid.yahoo.com
  - ./zkCli.sh -server fubariteblue-ni-zk0.blue.ygrid.yahoo.com:50512

```
ls /
[mobileApp, transactional, storm, nrtcf, zookeeper]
ls /nrtcf
[partition_3, partition_4, partition_1, partition_2,
 partition_10, partition_11, partition_0, partition_12,
 partition_13, partition_14, partition_9, partition_6,
 partition_5, partition_8, partition_7]

get /nrtcf/partition_1
{"topology":{"id":"028110a5-5386-4512-b9c1-2fec8957815c","name":"nrtcf"},
 "offset":24044930,"partition":1,
  "broker":{"host":"kafka3.dp.cb.gq1.yahoo.com","port":9092},
  "topic":"SearchTraffic"}

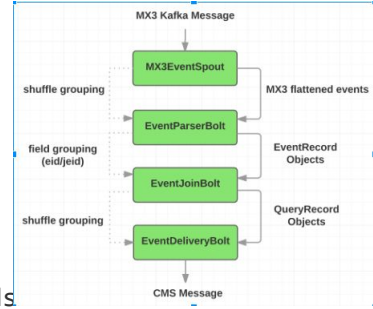
...
```

Tuple Transfer



- A generic task is composed of 2 threads and 2 queues
- Inter-worker communication
  - Inter-thread on the same node - LMAX disruptor
  - Inter-worker communication (node-to-node) - ZeroMQ/Netty

EventParser Bolt



- Subscribes to Kafka Spout receiving `SerializedSearchTrafficRecordList records`
  - https://git.corp.yahoo.com/Native-Ads/mx3-llp-common/blob/master/src/main/java/com/yahoo/gemini/llp/common/schema/pb/SearchTraffic.proto
  - https://git.corp.yahoo.com/Curveball/curveball_nrt_click_feedback/blob/master/storm-app/src/main/java/com/yahoo/curveball/nrtcf/stormapp/records/SearchRecordFilter.java
  - Unpack SearchTrafficRecords (contains SM_SERVE, SM_MERGE and SM_CLICK events)
  - Validate various fields bcookie, join_event_guid, page position etc in SearchTrafficRecord
    - Filter no ads serve events
  - Finally construct FilteredSearchRecords object containing valid SearchTrafficRecords
- Emitting tuples
  - Emits stream containing tuple with two fields (event_id, SearchTrafficRecord)
  - Pay attention to event_id (use join_event_guid for SM_MERGE and SM_CLICK)
    - https://git.corp.yahoo.com/Curveball/curveball_nrt_click_feedback/blob/master/storm-app/src/main/java/com/yahoo/curveball/nrtcf/stormapp/bolts/EventParserBolt.java

```
public void declareOutputFields(OutputFieldsDeclarer outputFieldsDeclarer) {
    outputFieldsDeclarer.declareStream(Constants.EventParserBolt.STREAM, new Fields(
            Constants.EventParserBolt.EVENT_ID, Constants.EventParserBolt.SEARCH_TRAFFIC_RECORD));
}
```

EventJoinBolt

- Subscribes SearchTrafficRecords from EventParserBolt
  - Uses fieldsGrouping on event_id to receive SM_SERVE, SM_MERGE and SM_CLICK events belonging to same search event landing on same EventJoinBolt instance
  - SM_SERVE event contains ads served by Gemini
  - SM_MERGE contains ads shown to the user (mix of ads from Gemini and Bing)
    - Some ads may be dropped or deduped depending on HM algo
  - https://git.corp.yahoo.com/Curveball/curveball_nrt_click_feedback/blob/master/storm-app/src/main/java/com/yahoo/curveball/nrtcf/stormapp/bolts/EventJoinBolt.java
- Join strategies
  - Join SM_SERVE with SM_MERGE events
  - Update click information in join events based on SM_CLICK events
  - These events can be arrived out of order - forces to buffer events
  - Clicks usually arrive after serve and merge events

EventJoinBolt Output Tuple

```
message QueryRecord {

        required int64 stim = 1; // SMServe event receive time
        required string dev = 2; // device type id
        required string rawq = 3; // raw user query
        optional string vcq = 4; // variant canon query
        repeated Impression ads = 5; // list of North Ads impressions

        message Impression {
                required string ppos = 1; // page position after merge
                optional int32 ltype = 2; // Ad listing type: 1-Yahoo, 2-Bing
                optional int32 clk = 3; // click count
                required string url = 4; // Ad display url
                optional string title = 5; // Ad title
                optional string desc = 6; // Ad description
        }

}
```

- Emits a tuple containing two fields bcookie and all north impressions shown for a given search query (QueryRecord)
  - https://git.corp.yahoo.com/Curveball/curveball_nrt_click_feedback/blob/master/common/src/protobufs/query_record.proto
  - https://git.corp.yahoo.com/Curveball/curveball_nrt_click_feedback/blob/master/storm-app/src/main/java/com/yahoo/curveball/nrtcf/stormapp/bolts/EventJoinBolt.java
  - QUERY_RECORD_KEY is bcookie of the search event

```
public void declareOutputFields(OutputFieldsDeclarer declarer) {
    declarer.declareStream(Constants.EventJoinBolt.STREAM, new Fields(Constants.EventJoinBolt.QUERY_RECORD_KEY,
        Constants.EventJoinBolt.QUERY_RECORD));
}
```

Time Slice Queue

```
public class TimeSliceQueue<T> {

    private int size;
    private int resolution;
    private long startTimestamp;
    private int startIndex = 0;
    private List<T> list;

}
```

- Data Structure to buffer SM_SERVE/SM_MERGE/SM_CLICK events
  - Represents fixed time duration (window) sliced into several time slots with equivalent widths
  - Window can move forward by slots when new record arrives, old records out of duration window will be dropped
    - https://git.corp.yahoo.com/Curveball/curveball_nrt_click_feedback/blob/master/storm-app/src/main/java/com/yahoo/curveball/nrtcf/stormapp/common/TimeSliceQueue.java
- EventJoinBolt maintains 3 Time Slice Queues (Serve, Merge and Clicks)
  - Need to cache merge and click events if they arrive before serve events

```
private static final int DEFAULT_QUERYRECORD_QUEUE_SIZE = 60;   // 60 slots
private static final int DEFAULT_QUERYRECORD_QUEUE_RESOLUTION = 60;   // duration for each slot is 60 secs

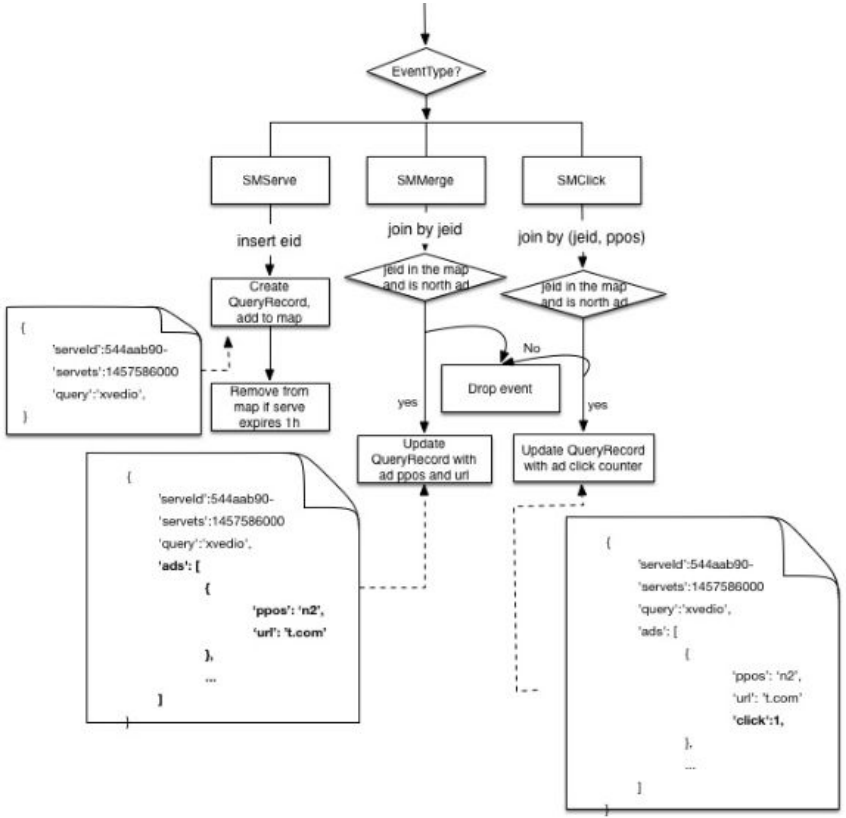
private static final int DEFAULT_MERGE_QUEUE_SIZE = 5;   // 5 slots
private static final int DEFAULT_MERGE_QUEUE_RESOLUTION = 60;   // duration for each slot is 60 secs

private static final int DEFAULT_CLICK_QUEUE_SIZE = 20;   // 20 slots
private static final int DEFAULT_CLICK_QUEUE_RESOLUTION = 60;   // duration for each slot is 60 secs

private TimeSliceQueue<Map<String, QueryRecord>> queryRecordsQueue;   // Serve Events
private TimeSliceQueue<Map<String, List<Impression>>> mergeImpsQueue;
private TimeSliceQueue<Map<String, List<SearchTrafficRecord>>> clickEventsQueue;
```

Event Join Logic

- https://docs.google.com/document/d/1j-KKO2RJcas_3hcHqRW6JlSyfkf-8jln467AMEzyzGU/edit?pli=1#
- QueryRecord captures query information (bcookie, raw query, device type and the set of north impressions associated with this query)
- One search query results in multiple SM_SERVE and SM_MERGE events (one for each ad)
- First SM_SERVE event for a given search creates QueryRecord and getting updated with impression when SM_MERGE event arrives for this search
- Each SM_SERVE/SM_MERGE event captures total number of ads for the search query (can be used to trigger the completion of one search query processing)
- If SM_MERGE/SM_CLICK events arrive before SM_SERVE, they are cached until SM_SERVE arrived



Emitting QueryRecords

```
message QueryRecord {

        required int64 stim = 1; // SMServe event receive time
        required string dev = 2; // device type id
        required string rawq = 3; // raw user query
        optional string vcq = 4; // variant canon query
        repeated Impression ads = 5; // list of North Ads impressions

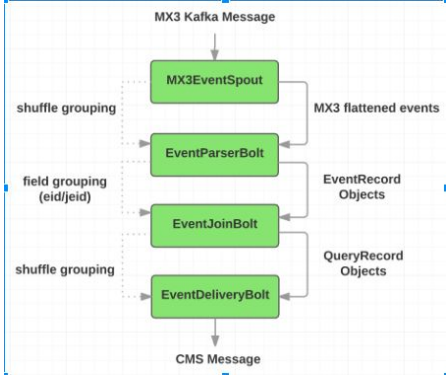
        message Impression {
                required string ppos = 1; // page position after merge
                optional int32 ltype = 2; // Ad listing type: 1-Yahoo, 2-Bing
                optional int32 clk = 3; // click count
                required string url = 4; // Ad display url
                optional string title = 5; // Ad title
                optional string desc = 6; // Ad description
        }

}
```

- Accumulates processed QueryRecords
  - Processing of QueryRecord is completed once all of its serve and merge events received
  - Emits QueryRecords at a fixed interval (every second)
  - May or may not have click information
- When a click arrives, query record gets updatec
  - Re-emits that query record again
  - Please note that query records are cached for an hour
- Why did we choose CMS instead of another Kafka Topic?
  - Bad reason: Don’t want to maintain kafka cluster (may not be good reason)
  - One end is kafka and another end is CMS

```
this.emittingTask = new TimerTask() {
    @Override
    public void run() {
        try {
            emitPooledQueryRecords();
        } catch (Throwable e) {
            Logger.error("emitting task failed", e);
        }
    }
};
this.emittingTimer = new Timer();
this.emittingTimer.scheduleAtFixedRate(this.emittingTask, this.emittingInterval, this.emittingInterval);
```

EventDelivery Bolt



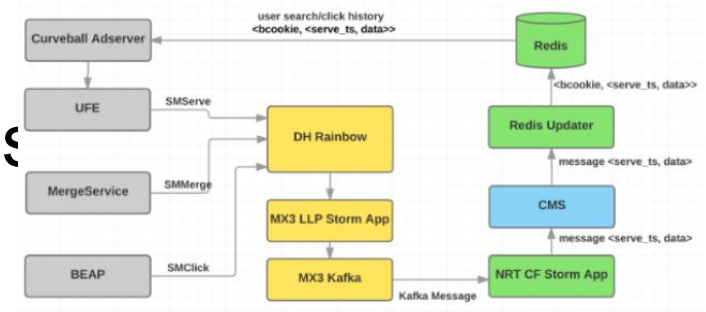
- Sink bolt - subscribes to EventJoinBolt (ShuffleGrouping)
- Batches QueryRecords before committing to CMS
  - https://git.corp.yahoo.com/Curveball/curveball_nrt_click_feedback/blob/master/storm-app/src/main/java/com/yahoo/curveball/nrtcf/stormapp/bolts/EventDeliveryBolt.java
  - https://git.corp.yahoo.com/Curveball/curveball_nrt_click_feedback/blob/master/common/src/protobufs/assemblies.proto
- CMS Message
  - Currently each Assembly record contains only one QueryRecord for a bcookie
  - If there are multiple searches for a given bcookie, they are emitted as individual records

```
message Assemblies {
        required int64 tim = 1;
        repeated Assembly msgs = 2;

        message Assembly {
                required string bck = 1; // bcookie
                repeated QueryRecord searches = 2; // list of QueryRecords; will only contain 1 given current implementation;
        }

}
```

NearRealTime User Clkb Features



- Compute real time coec (clicks over expected clicks) for a given user for the following features
  - User, [User, Query], [User, Ad Domain]
  - https://git.corp.yahoo.com/Curveball/adserver/blob/curveball.adserver.1.44.96.nrtcf.branch/modules/NRTCFFeatureModule/src/NRTCFFeatureModule.cc
  - User Query Records are pushed to redis cluster (from CMS to Redis)
  - Ad Server interacts with redis to fetch all QueryRecords for a given user in the last 48 hours
- Aggregate metrics (ec, coec) from Query Records
  - Query level, Ad Domain Level
  - Expected clicks is computed based on [device type, position] ref CTR
  - Aggregation is happening for every request (not a great idea)

Deployment

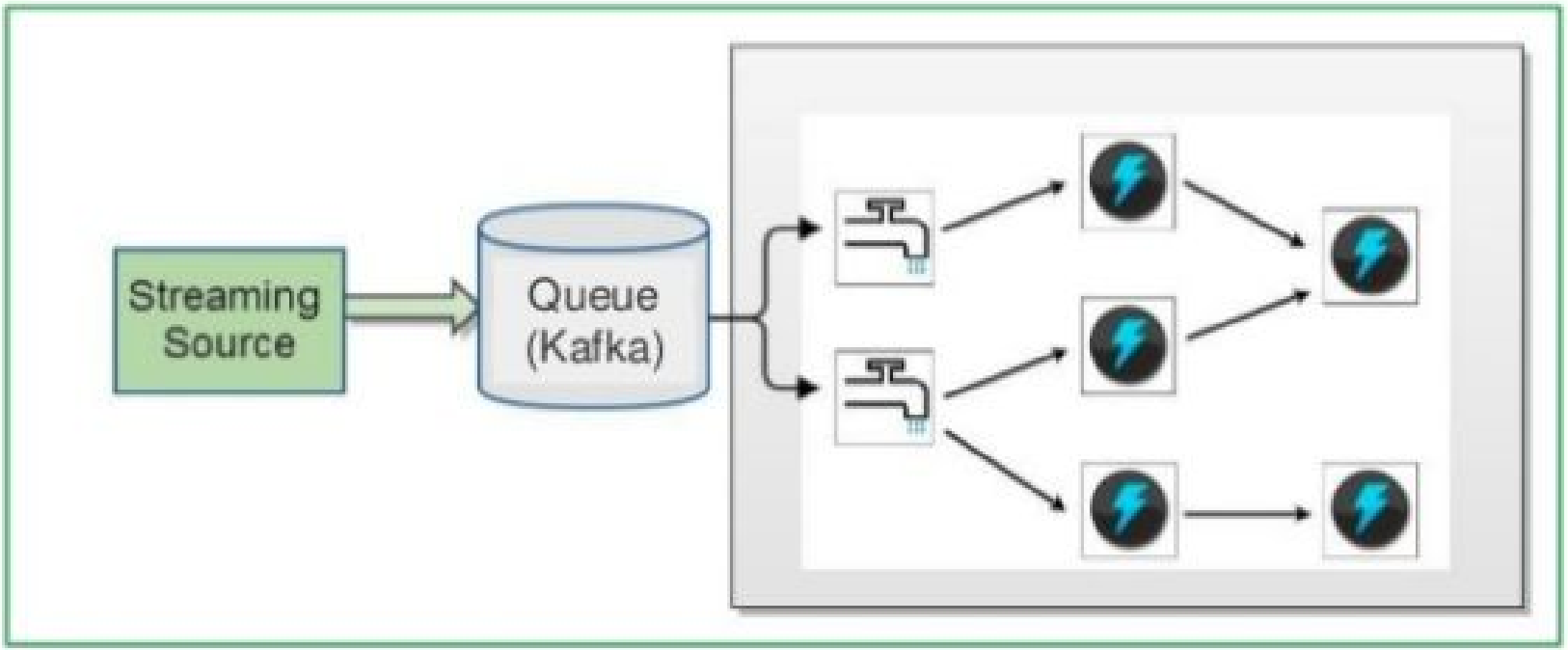
- Deployment host: cf-launcher1.cbs.cb.gq1.yahoo.com
  - -bash-4.1$ yroot fb 6.5-20160307

```
curveball_nrtcf_storm_app.event_click_queue_resolution: 60
curveball_nrtcf_storm_app.event_click_queue_size: 20
curveball_nrtcf_storm_app.event_delivery_bolt_cpu: 100
curveball_nrtcf_storm_app.event_delivery_bolt_memory: 4096
curveball_nrtcf_storm_app.event_join_bolt_cpu: 6
curveball_nrtcf_storm_app.event_join_bolt_memory: 8192
curveball_nrtcf_storm_app.event_join_bolt_parallelism: 2
curveball_nrtcf_storm_app.event_merge_queue_resolution: 60
curveball_nrtcf_storm_app.event_merge_queue_size: 5
curveball_nrtcf_storm_app.event_parser_bolt_cpu: 100
curveball_nrtcf_storm_app.event_parser_bolt_memory: 4096
curveball_nrtcf_storm_app.event_parser_bolt_parallelism: 1
curveball_nrtcf_storm_app.event_queryrecord_flush_interval: 1000
curveball_nrtcf_storm_app.event_queryrecord_queue_resolution: 60
curveball_nrtcf_storm_app.event_queryrecord_queue_size: 60
curveball_nrtcf_storm_app.kafka_spout_cpu: 100
curveball_nrtcf_storm_app.kafka_spout_memory: 4096
curveball_nrtcf_storm_app.kafka_spout_parallelism: 1
curveball_nrtcf_storm_app.mx3llp_consumer_group_id: nrtcf
curveball_nrtcf_storm_app.mx3llp_consumer_topic: SearchTraffic
```

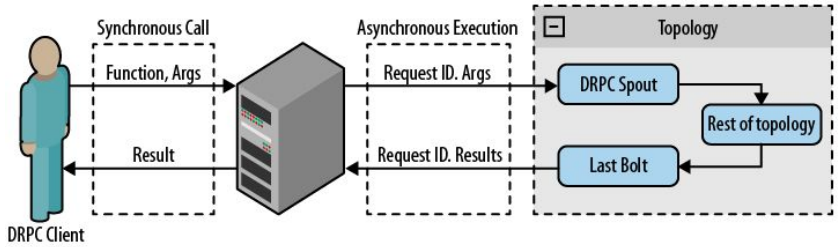
```
ystorm.nimbus_host: fubariteblue-ni.blue.ygrid.yahoo.com
ystorm.nimbus_thrift_port: 50560
ystorm.storm_zookeeper_port: 50512
ystorm.storm_zookeeper_root: /storm/fubariteblue-ni
ystorm.storm_zookeeper_servers: fubariteblue-ni-zk0.blue.ygrid.yahoo.com,
    fubariteblue-ni-zk1.blue.ygrid.yahoo.com,
    fubariteblue-ni-zk2.blue.ygrid.yahoo.com,
    fubariteblue-ni-zk3.blue.ygrid.yahoo.com,
    fubariteblue-ni-zk4.blue.ygrid.yahoo.com
```

YMAS Metrics Dashboard

Storm Architectural Blueprint



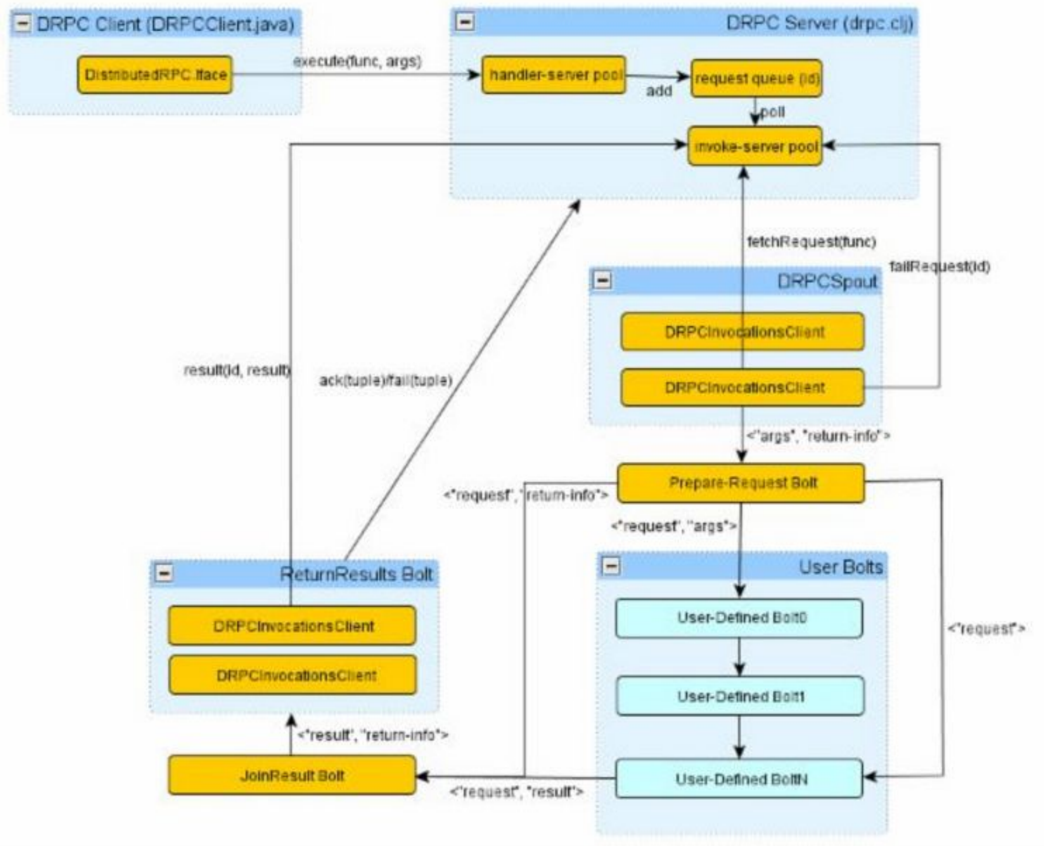
DRPC Topology



- Distributed Remote Procedure Call (DRPC)
  - Executes RPC using distributed power of storm
  - DRPC server runs as a connector between client and topology
    - Runs as source for the topology
    - Receives the function to execute and its arguments
    - Generates unique request id for the function to be executed
    - When the last bolt of topology finishes execution, it returns result to DRPC server
  - Provides `LinearDRPCTopologyBuilder` to create DRPC topologies

DRPC Architecture - DRPC Server

- DRPC Server
- Spouts
  - DRPCSpout
- Bolts
  - Prepare-request Bolt
  - JoinResultBolt
  - ReturnResultsBolt

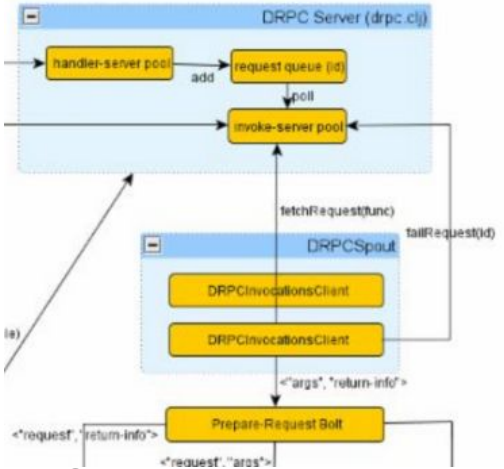


DRPCServer

drpc.http.port	4080
drpc.https.keystore.password	""
drpc.https.keystore.type	"JKS"
drpc.https.port	-1
drpc.invocations.port	50571
drpc.invocations.threads	64
drpc.max_buffer_size	1048576
drpc.port	50570
drpc.queue.size	128
drpc.request.timeout.secs	600
drpc.servers	["gsrd454n00.red.ygrid.yahoo.com", "gsrd455n00.red.ygrid.yahoo.com", "gsrd456n00.red.ygrid.yahoo.com"]
drpc.worker.threads	64

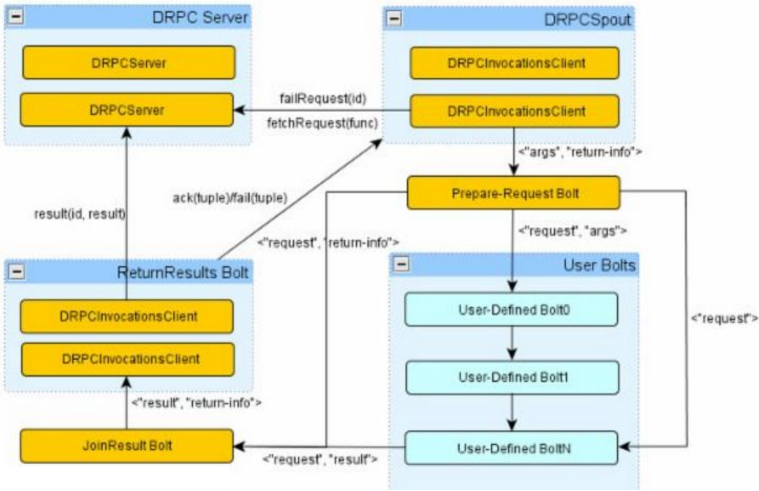
- DRPC Server
  - Each storm cluster is configured to run a couple of DRPC Servers
  - User can send request to DRPC Server (simple http request)
  - https://github.com/apache/storm/blob/master/storm-core/src/jvm/org/apache/storm/daemon/DrpcServer.java

DRPCSpout



- Fetches request from DRPC Server
  - Reads DRPC Server hosts/ports from configuration
  - Maintains cached connections to DRPC Servers
  - nextTuple() tries to fetch request from DRPC servers
    - Depending on parallelism, it may fetch requests one or more DRPC servers
    - DRPCSpout specifies the function_name (what types of requests it wants to process)
  - Emits a tuple containing “args of the request” and “return-info” for the request
    - Return-info contains DRPC host/port which initiated the request
  - https://github.com/apache/storm/blob/master/storm-core/src/jvm/org/apache/storm/drpc/DRPCSpout.java

PrepareRequest Bolt



- Receives request from DRPC Spout containing “args of the request” and “return-info” for the request

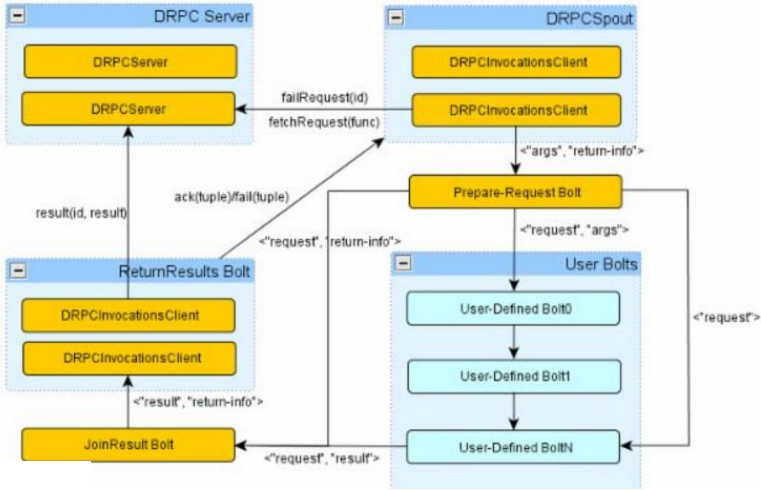
```
SpoutDeclarer drpcSpout = builder.setSpout(SPOUT_ID, new DRPCSpout(function));
BoltDeclarer prepareRequest = builder.setBolt(PREPARE_ID, new PrepareRequest()).noneGrouping(SPOUT_ID);
```

- Generates new request id
- Creates one stream containing “request” and “args” (processing bolts subscribe to this stream)
- Creates another stream containing “request” and “return-info” (JoinResultsBolt subscribes to it)
  - https://github.com/apache/storm/blob/master/storm-core/src/jvm/org/apache/storm/drpc/PrepareRequest.java

```
@Override
public void execute(Tuple tuple, BasicOutputCollector collector) {
    String args = tuple.getString(0);
    String returnInfo = tuple.getString(1);
    long requestId = rand.nextLong();
    collector.emit(ARGS_STREAM, new Values(requestId, args));
    collector.emit(RETURN_STREAM, new Values(requestId, returnInfo));
    collector.emit(ID_STREAM, new Values(requestId));
}

public void declareOutputFields(OutputFieldsDeclarer declarer) {
    declarer.declareStream(ARGS_STREAM, new Fields("request", "args"));
    declarer.declareStream(RETURN_STREAM, new Fields("request", "return"));
    declarer.declareStream(ID_STREAM, new Fields("request"));
}
```

JoinResult Bolt



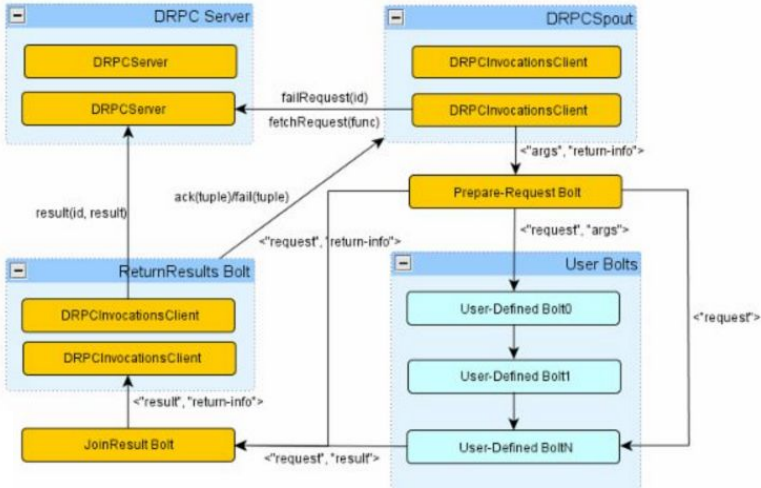
- Collects “return-info” (DRPC host, port) for a given request from “prepare-request” bolt
- Similarly collects “result” from the last bolt.

```
BoltDeclarer joinResult = builder.setBolt("JoinResult", new JoinResult(PREPARE_ID))
        .fieldsGrouping("ResultCollector", "result", new Fields("request"))
        .fieldsGrouping(PREPARE_ID, PrepareRequest.RETURN_STREAM, new Fields("request"));
```

- Finally JoinResult Bolt joins “request, result” and “request, return-info” and returns “result, return-info” to ReturnResultsBolt
  - Just maintains two maps - one for results and one for return-info (both keyed by request-id)
  - Based on incoming tuple (if it is coming “prepare-request” bolt, it keeps that info in returns map), otherwise it stores it in “results” map. Once it has both entries in results and returns maps, emit tuple (results, return-info) to ReturnResultsBolt
  - https://github.com/apache/storm/blob/master/storm-core/src/jvm/org/apache/storm/drpc/JoinResult.java

ReturnResults Bolt

- Return results to DRPC server which initiated the request




```
BoltDeclarer joinResult = builder.setBolt("JoinResult", new JoinResult(PREPARE_ID))
        .fieldsGrouping("ResultCollector", "result", new Fields("request"))
        .fieldsGrouping(PREPARE_ID, PrepareRequest.RETURN_STREAM, new Fields("request"));

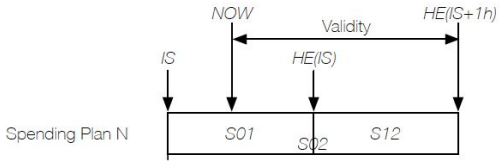
BoltDeclarer returnResults = builder.setBolt("ReturnResults", new ReturnResults()).noneGrouping("JoinResult");
```

- “Return-info” contains host and port of DRPC server initiated the request
  - Uses DRPCInvocationsClinet to connect to DRPC server and delivers result to DRPC server
  - “Return-info” also contains unique request id generated by DRPC server - client.result(id, result);
  - https://github.com/apache/storm/blob/master/storm-core/src/jvm/org/apache/storm/drpc/ReturnResults.java
  - Also acks result with DRPCSpout through Acker bolt.

Gemini Budget

Spend Management in Gemini

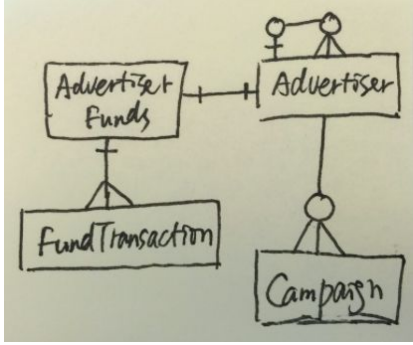
- Goal: Control servable statuses of advertisers and campaigns based on delivery rules
- What are delivery rules ?
  - Spend Cap Type - Daily, Monthly, Total etc
  - Spend Cap Amount - Available budget for the given spend cap type
  - Pacing Type - ASAP and EVEN (deliver budget smoothly)
  - Start and end time
- What is spend plan?
  - Contains the budget constraints that cover up to 2 hours from spending plan start time
    - Interval start (IS)
    - Budget Constraint S01 - Budget from IS to the hour end of IS, referred as HE(IS)
    - Budget Constraint S02 - Budget from IS to the second hour end of IS: HE(IS + 1h)
    - Budget Constriant S12 - Available budget from HE(IS) to HS(IS + 1h)



Spending Plan Generation

- What do we need to generate a Spending Plan?
  - Delivery rules
  - Historical spend that covers until interval start time (IS)
    - Total spend starting from the start of this hour
    - Total spend starting from the start of this day (honor timezone aspects of advertiser)
    - Total spend starting from the start of this month
    - Total spend starting from the beginning of its life
- Where can I find delivery rules ?
  - Available daily snapshots and incremental snapshots on HDFS (json format)
    - COW snapshots (Curveball Operational Warehouse)
    - Also available in HBASE tables
    - Or can consume delivery rule object changes from CMS directly
- Historical Spend
  - Historical spend feeds available on HDFS (every 15 minutes, TP filtered)

Delivery Rules



- Advertiser - Represents a demand customer, groups campaigns together
- AdvertiserFunds - Specifies the advertisers payment plan, available funds and balance owed
- FundTransactions - Represents an advertisers financial transaction (credit/debit)
- Campaigns - Grouping of ad groups that share budget
    - HBASE: http://luxblue-hb.blue.ygrid.yahoo.com:50500/tablesDetailed.jsp
        - cow_prod:PROD_COW_ADVERTISER (Key: Advertiser Id)
        - cow_prod:PROD_COW_CAMPAIGN (Key: Advertiser Id, Campaign Id)
        - cow_prod:PROD_COW_DOMAINOBJECTS (advertiser_funds, fund_transactions) - Key (Domain Object Type, Object Id)
            - Only one column d: contains JSON object describing all fields.
        - Example: (Fetch last 2 versions of advertiser records for id: 1000117)
            - ssh phazon-gw.blue.ygrid.yahoo.com (do kinit with credentials)
            - hbase shell
            - hbase(main):001:0> get 'cow_prod:PROD_COW_ADVERTISER', 1000117, {COLUMN =>'d:m', VERSIONS=>2}
    - COW Snapshots locations (PB: ssh phazon-gw.blue.ygrid.yahoo.com)
        - Daily Snapshots: hadoop dfs -ls /projects/cow_prod/complete_snapshot/<day>
        - Incremental Snapshots: hadoop dfs -ls /projects/cow_prod/incremental_snapshot/<date> (every 15 min)

Advertiser Level Delivery Rules

- https://docs.google.com/document/d/1EF6_jRIaTWoT2PuS4B6LSSBJZjWKuPKDxSuBKONEXtM/edit#

#	paymentPlan	funds	Amount	accountSpendCapType	startDate	endDate	Behavior
1	POSTPAY	ignored	X	MONTHLY	not null	not null	ASAP / from start date to end date / monthly cap X
2	POSTPAY	ignored	ignored	NONE	null or not null	null or not null	ASAP / from start date to end date (if has) / unlimited budget
3	POSTPAY	ignored	X	TOTAL	not null	not null	ASAP / from start date to end date / total cap X
4	PREPAY	Y	ignored	ignored	ignored	ignored	ASAP
5	PAYG	ignored	X	PAYG_THRESHOLD	not null	null	ASAP / from start date to end date / total cap X

Campaign Level Delivery Rules

#	budgetType	budget	spendCap	spendCapType	pacingType	Behavior
1	UNLIMITED	ignored	Z	DAILY	ASAP	Daily ASAP / daily cap Z / unlimited lifetime cap
2	UNLIMITED	ignored	Z	DAILY	EVEN / DEFAULT / NULL	Daily EVEN / daily cap Z / unlilmited lifetime cap
3	UNLIMITED	ignored	ignored	NONE	ignored	Illegal combination A campaign cannot have nothing but an unlimited cap (a campaign must have a cap, either lifetime or daily)
4	LIFETIME	ignored	ignored	DAILY	ignored	Illegal combination Note: A campaign cannot have both a lifetime cap and a daily cap. But such a combination is possible with the effective* fields.
5	LIFETIME	Y	ignored	NONE	ASAP	Lifetime ASAP / lifetime cap Y
6	LIFETIME	Y	ignored	NONE	EVEN / DEFAULT / NULL	Normalized Daily Cap, see details

Resellers

- Reseller Advertisers can create multiple advertiser accounts
  - Known as Managed advertisers
  - Spend is aggregated to Reseller level
  - Budget is at reseller advertiser level

```
SQL> select type, count(*) from mb.advertiser where status = 'ON' group by type;

TYPE            COUNT(*)
------------------------
ADVERTISER      318482
RESELLER        435
```

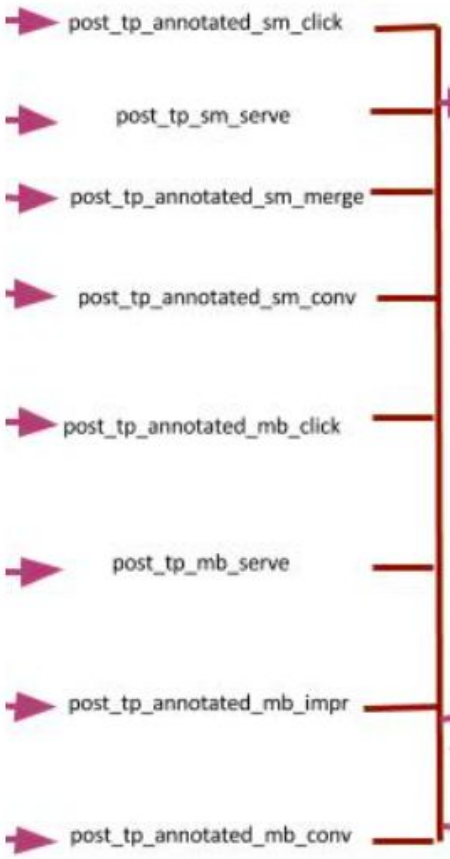
- Some resellers are having a lot of managed advertisers
  - For managed advertisers, advertsier's timezone is reseller's timezone
  - Spend/Imps/clicks are also aggregated at reseller level
    - https://git.corp.yahoo.com/MX3/mx3_spend_mgmt/blob/master/src/raw_spend_traffic.pig
    - https://git.corp.yahoo.com/budget/curveball/blob/master/devel/src/spend_plan_gen/src/main/java/com/yahoo/curveball/budget/common/beans/SpendPlanManagedInfo.java

```
SQL> select managed_by, count(*) as cnt from mb.advertiser where status = 'ON'
group by managed_by order by cnt DESC

MANAGED_BY    CNT
---------- ----------
45820   89426
1032812 10637
1034581  6659
1101208  5169
1056902  3504
1240659  3202
65604    2384
1166007  1889
1060973  1011
1029281   984
```

Historical Spend

- Spend amount aggregated every 15 min (mx3 data pipeline)
  - Input:
    - sm - search monetization events
    - mb - moneyball events (aka native serving)
  - Output
    - PB Path: /projects/mx3/prod/feeds/spend_mgmt_fast/15m/ (fast path)
    - PB Path: /projects/mx3/prod/feeds/spend_mgmt/15m/
- Spend Management Computation
  - Aggregate imps, clicks, conversions, serves by (advertiser id, campaign id, product type, test flag)
  - Also join with dim_io_budget to get current io and io_line_id for advertiser
    - https://git.corp.yahoo.com/MX3/mx3_spend_mgmt/blob/master/src/raw_spend_traffic.pig
  - Join with prev raw_spend to compute cumulative sums (full join)
    - https://git.corp.yahoo.com/MX3/mx3_spend_mgmt/blob/master/src/raw_spend.pig
  - Compute monthly, lifetime spend amounts
    - Adjust any credits if present
    - https://git.corp.yahoo.com/MX3/mx3_spend_mgmt/blob/master/src/spend_mgmt.pig



Historical Spend - Hive Interface

```
hive> desc mx3.spend_mgmt;
OK
batchdate               bigint
advertiser_id           bigint
campaign_id             bigint
product_type            string
timezone                string
test_flag               int
eod                     int
eom                     int
impressions             bigint
clicks                  bigint
conversions             bigint
hourly_revenue          string
daily_revenue           string
monthly_revenue         string
lifetime_revenue        string
serves                  bigint
io_id                   string
io_line_id              string
invalid_hourly_revenue  string
invalid_daily_revenue   string
invalid_monthly_revenue string
invalid_lifetime_revenue        string
feed_src                int
datestamp               string
```

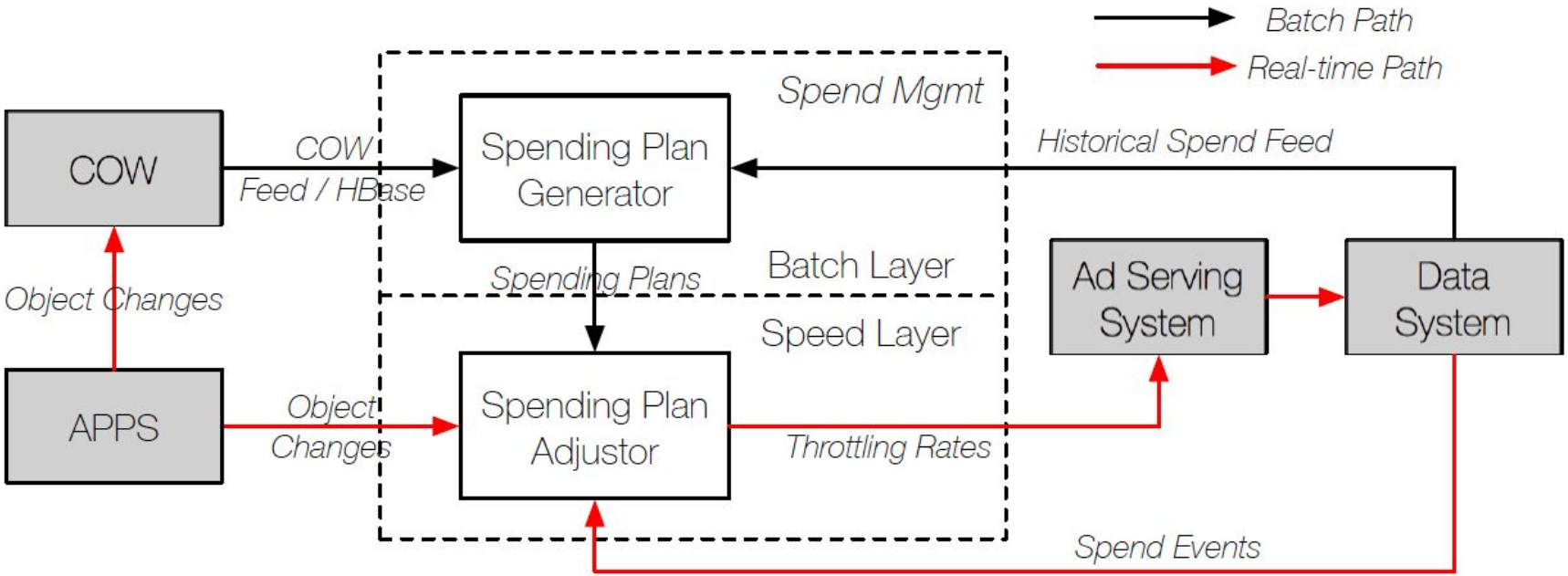
- Historical spend (spend_mgmt feed) is also available as external table in hive
  - location:hdfs://phazonblue-nn1.blue.ygrid.yahoo.com:8020/projects/mx3/prod/feeds/spend_mgmt/15m/dat
  - Partitioned by 15m interval datestamp
- Records for end of day/month are marked with eod/eom flags
  - eod/eom are marked according advertiser timezone
  - hourly/daily/monthly revenue are cumulative revenues - reset at boundaries
  - Lifetime_revenue is at [io_id, io_line_id] level
  - CAUTION: Contains all campaigns (active, inactive, completed)
- Here is sample query to identify hourly revenue group by product_type
  - Product_type - native/search
  - hive> select datestamp, product_type as pt, count(distinct advertiser_id) as adv_cnt, count(distinct campaign_id) as cmp_cnt, round(sum(hourly_revenue), 1) as hourly_revenue from spend_mgmt **where datestamp >= '201607130900' and datestamp <= '201607130945' and hourly_revenue > 0 and test_flag = 0** group by product_type,
  - Spend plan considers historical spend at advertiser/campaign level

datestamp	pt	adv_cnt	cmp_cnt	hourly_revenue
201607130900	native	7000	10970	228436.8
201607130915	native	7918	13137	366861.1
201607130930	native	8371	14260	391866.1
201607130945	native	8762	15218	511882.5
201607130900	search	1271	2464	5065.8
201607130915	search	1821	4199	10095.3
201607130930	search	2183	5670	16188.6
201607130945	search	2481	7041	23079.3

Fast Historical Spend

- Fast Historical Spend at datestamp Y
  - Contains spend_mgmt up to datestamp X + real time feed between datestamp {X and Y}
  - Example: /projects/mx3/prod/feeds/spend_mgmt_fast/15m/data/201607181800
    - Two kinds of records (identified by feed_src)
      - feed_src:0 - spend_mgmt feed (contains datestamp X)
        - It contains cumulative stats up to X+15min
      - feed_src:1 - Real time feed between [X+15 and Y]
    - There are two files, each one is capturing different feed
      - In this example, it has spend_mgmt feed utp 201607181630 + 15 min (feed_src: 0)
      - And real time feed between 201607181645 and 201607181800 (feed_src: 1)

Gemini Budget - Lambda Architecture



Spend Plan Example

- Select an advertiser with a couple of active campaigns from speng_mgmt
  - hive> select advertiser_id, campaign_id, product_type, round(hourly_revenue,1), round(daily_revenue,1), round(monthly_revenue,1), round(lifetime_revenue,1), io_id, io_line_id from spend_mgmt where datestamp='201607141630' and advertiser_id = 1417237 and test_flag = 0 order by campaign_id;

advertiser_i	campaign_id	product_typ	hourly_rev	daily_rev	monthly_re	lifetime_rev	io_id	io_line_id
1417237	-9999	native	$868	$5,064	$60,049	$60,049	a2N32000002iGFCEA2	a3N32000001iWaxEAE
1417237	-9999	native	$0	$0	$20	$18,435	a2N32000002iGFCEA2	a3N32000001i3vTEAQ
1417237	351036251	native	$0	$0	$18	$9,265	a2N32000002iGFCEA2	a3N32000001i3vTEAQ
1417237	351036251	native	$439	$2,444	$29,919	$29,919	a2N32000002iGFCEA2	a3N32000001iWaxEAE
1417237	351042292	native	$0	$0	$2	$9,170	a2N32000002iGFCEA2	a3N32000001i3vTEAQ
1417237	351042292	native	$429	$2,620	$30,130	$30,130	a2N32000002iGFCEA2	a3N32000001iWaxEAE

- SQL> select id, advertiser_id, start_time, end_time, budget, budget_type, spend_cap, spend_cap_type from mb.campaign where id = 351042292; (ORACLE DB)

campaign_id	advertiser_id	start_time	end_time	budget	budget_type	spend_cap	spend_cap_type
351042292	1417237	28-JUN-16 07.00.00 AM	16-JUL-16 06.59.59 AM	$55,000	LIFETIME	NONE	
351036251	1417237	28-JUN-16 07.00.00 AM	16-JUL-16 06.59.59 AM	$55,000	LIFETIME	NONE	

Spend Plan Example - Advertiser Funds

- Advertiser Funds
  - select advertiser_id, funds, balance, payment_plan, account_spend_cap, account_spend_cap_type from mb.advertiser_funds where advertiser_id = 1417237;

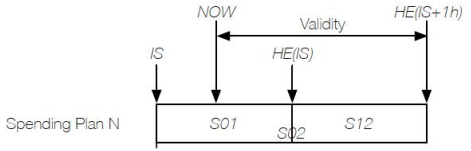
advertiser_id	funds	balance	payment_plan	account_spendcap	account_spendcap type
1417237	$0	$0	POSTPAY	$0	TOTAL

- Advertiser Fund Transactions
  - SQL> select id, amount, cap_type, io_id, io_line_id, start_date, end_date from mb.fund_transactions where advertiser_id = 1417237;
  - Only one IO is active at a time, Advertiser Fund Transactions overrides Advertiser Funds attributes.

id	amount	cap_type	io_id	io_line_id	start_date	end_date
1985022	$110,000	MONTHLY	a2N32000002iGFCEA2	a3N32000001i3vTEAQ	06-JUN-16 07.00.00 AM	01-JUL-16 06.59.59 AM
2059141	$91,585	TOTAL	a2N32000002iGFCEA2	a3N32000001iWaxEAE	11-JUL-16 07.00.00 AM	16-JUL-16 06.59.59 AM

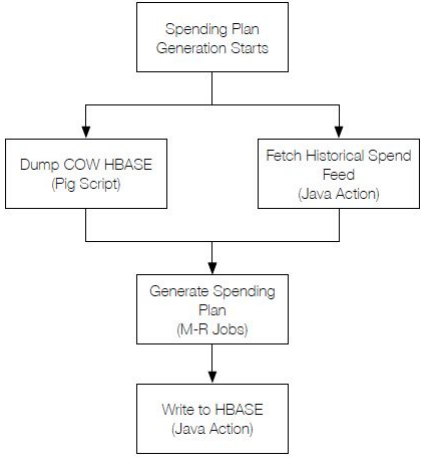
Generated Spend Plan - Campaigns

- Computes available budget for the next 2 hours
  - Sample output: /projects/cb_budget/spgen_grid/spgen_grid_output/spgen_output/
  - Campaigns spend plan
    - Computes budget for the next two hours (S01, S02 and S12)
      - Depends on spend_cap, pacing_type and amount spent
    - Here are couple of examples -
      - Daily/Default (pacing_type - even)
      - Compute remaining time and available budget (daily cap - amount spent)
      - Now compute budget for S01 as remaining time in that hour * (available budget/remaining time)
      - Similarly compute budget for S02 (first hour + second hour)
      - Pay attention if S12 is falling into second day (day boundary) then adjust budget accordingly
        - https://docs.google.com/document/d/1VXhSzIDu97yeS77WkQLqO-PMq8doG2He_7uLha7u4o8/edit#



c.id	c.budget	c.budget_type	c.spend_cap	c.spend_cap_ty	c.pacing_type	c.start_time	c.end_time	c.daily_spend	c.interval_start	c.plan_gen_start	c.s01	c.s02	c.s12
351434122	1.00E+20	UNLIMITED	$100,000	DAILY	DEFAULT	201607080400	-1	$11,877	201607200025	201607200045	$15,403	$42,447	1.00E+20
337448304	1.00E+20	UNLIMITED	$10,000	DAILY	ASAP	201411120500	-1	$4,628	201607200025	201607200045	1.00E+20	$5,372	1.00E+20
350760172	100000	LIFETIME	0	NONE	DEFAULT	201606210400	201607220359	4200.288	201607200025	201607200045	1264.8816	2911.7437	1.00E+20
340559307	150000	LIFETIME	0	NONE	ASAP	201505070400	201701010459	358.64	201607200025	201607200045	1.00E+20	63348.605	1.00E+20

Spend Plan Generation workflow

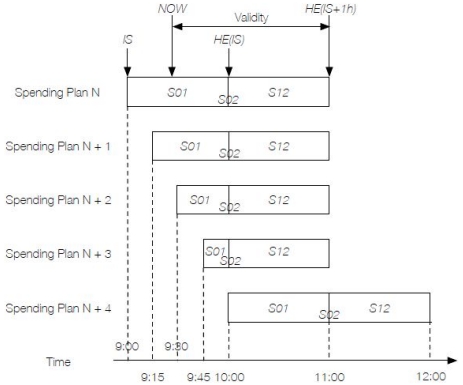


- Spend plan generation oozie workflow
  - https://git.corp.yahoo.com/budget/curveball/blob/master/devel/src/spgen_grid/src/main/oozie/workfl
  - http://phazonblue-oozie.blue.ygrid.yahoo.com:4080/oozie/
  - Use custom filter “name=Spgen-MapReduce-wf” to identify Spend plan generation WF jobs
- Workflow
  - Dump hbase tables advertiser, campaign
    - cow_prod:PROD_COW_ADVERTISER (536K records), cow_prod:PROD_COW_CAMPAIGN (12M records)
  - Dump advertiserFunds, fundTransactions objects from domain objects table
    - cow_prod:PROD_COW_DOMAINOBJECTS (531K advertiser_funds records, 1.3M fund_transactions records)
  - Get latest available fast spend_mgmt feed and determine Interval start (up to what timestamp does fast spend includes cum spend)
  - Now run map/reduce job to compute spend plan for each of campaigns and advertisers
    - First aggregate spend info (hourly, daily, monthly and lifetime) at advertiser and campaign level
    - For each advertiser, compute spend plan (available budget for the next 2 hours) using advertiser spend_cap_type, funds and amount already spent
    - Similarly for each campaign, compute spend plan using campaign’s budget, budget_type, spend_cap, spend_cap_type, pacing_type, start/end dates and amount already spent
    - SpendPlan is only computed for active advertisers and campaigns (for non-active objects, throttle rate is set to 1.0)

Spend Plan Generation

- Given Advertiser (funds, spend cap) and amount spent so far, compute available budget for the next couple of hours.
- Given Campaign (budget, spend_cap_type, pacing_type) and amount spent so far, compute available budget for the next couple of hours
  - https://git.corp.yahoo.com/budget/curveball/blob/master/devel/src/spgen_grid/src/main/java/com/yahoo/curveball/budget/spgen/SpgenReducer.java
  - https://git.corp.yahoo.com/budget/curveball/blob/master/devel/src/spgen_grid/src/main/java/com/yahoo/curveball/budget/grid/spendplan/SpendPlanMRBuilder.java
    - builder.genSpendPlan(intervalStart, planGenStart, domainObject, spendInfo);
    - Create ISpendPlanner instance based on spend_cap_type, pacing_type etc (SpendPlanerFactory.java)
  - Different types of SpendPlanners based on spend_cap_type, pacing_type ...
    - https://git.corp.yahoo.com/budget/curveball/tree/master/devel/src/spend_plan_gen/src/main/java/com/yahoo/curveball/budget/generator
    - MonthlyEVENSpendPlaner, MonthlyASAPSpendPlaner, LifeASAPSpendPlaner ...
    - Each spend planner captures how to compute budget for the next 2 hours(s01, s02, s12)

Spend Plans



- Spend plans for advertisers/campaigns are generated every 15 min
  - HDFS Output - /projects/cb_budget/spgen_grid/spgen_grid_output/spgen_output/
  - They are also persisted to HBASE
  - Every cycle is generating 12M records (advertisers and campaigns)
    - We do generate spend plan record (throttling rate set to 1.0) for non-active campaigns/advertisers
      - 80% records are non-active
      - Active - 1.7M search campaigns, 150K native, 50K both and 300K advertisers
- HBASE Spend Plans Table - Budget:spending_plans
  - http://luxblue-hb.blue.ygrid.yahoo.com:50500/tablesDetailed.jsp
  - https://git.corp.yahoo.com/budget/curveball/blob/master/devel/src/spgen_grid/src/main/java/com/yahoo/curveball/budget/spgen/SpgenReducer.java (emitResultToHbase)
    - https://git.corp.yahoo.com/budget/curveball/blob/master/devel/src/spend_plan_gen/src/main/java/com/yahoo/curveball/budget/generator/SpendPlan.java
    - `CacheRowKey rowKey = new CacheRowKey(plan.getVersion(), plan.getCacheId(), plan.getId(), plan.getType());`

SpendingPlans HBASE record Format

- RowKey
  - partition - generate partition key (4 bytes)
  - version id
  - cache_id - Plan generation timestamp
  - object_type - (advertiser/campaign)
  - object_id - (advertiser_id or campaign_id)
- Value (one column family, two columns)
  - sp - Budget Info (s01, s02, s12, daily_cap, daily_spend, cap_type, pacing_type, timezone etc)
  - st - Throttling rate
  - sman (optional column)- contains managed advertiser name (reseller advertisers)
  - https://docs.google.com/document/d/1-mTBeNCsd2YBlSMVFS10ynKDjcAIscyfo77VHXg4QSA/edit#heading=h.rnyvb9602ajk
- Partition Information
  - https://git.corp.yahoo.com/budget/curveball/blob/master/devel/src/hbase/src/main/java/com/yahoo/curveball/budget/hbase/CacheRowKey.java
  - 2 digits day + 2 digits based on hash(obj_id + types) mod 25
  - Partition key range - [01-31][00-24]

budget:spending_plans

- Partitions = 31 (days) * 24 buckets
- SpendPlan is generated every 15 min
  - Contains approximately 12M objects (campaigns and advertisers 07/20/2016)
  - Each record is inserted with key as <day><hash of obj_id+type mod 24> + version + timestamp + object_id + type
  - Any given day, records are getting inserted into 24 partitions (first 2 digits is day)
  - Currently 8 region servers are hosting all 775 partitions

Table Regions

Name	Region Server	Start Key	End Key
budget:spending_plans,,1389648881357.014904581fa95f42980f20a7cd805f1f.	gsrd557n24.red.ygrid.yahoo.com:50511		0100
budget:spending_plans,0100,1389648881357.6c227c9498177745f96702a9ec6a9c28.	gsrd557n24.red.ygrid.yahoo.com:50511	0100	0101
budget:spending_plans,0101,1389648881357.c8373127b7ed3272b5e3cc57daaf6562.	gsrd561n23.red.ygrid.yahoo.com:50511	0101	0102
budget:spending_plans,0102,1389648881357.16b20760c0934c1a67fd715a56921382.	gsrd557n25.red.ygrid.yahoo.com:50511	0102	0103
budget:spending_plans,0103,1389648881357.4a1160a1f7dd5004117619a9fa5f6b70.	gsrd561n26.red.ygrid.yahoo.com:50511	0103	0104
budget:spending_plans,0104,1389648881357.3458c0e51ee51614d70921ca43dfa73e.	gsrd561n26.red.ygrid.yahoo.com:50511	0104	0105
budget:spending_plans,0105,1389648881358.19ccc8142f378f81752522d6190cedaf.	gsrd557n25.red.ygrid.yahoo.com:50511	0105	0106
budget:spending_plans,0106,1389648881358.29a53b3a2924d152c4df2a6d5c98b13d.	gsrd561n23.red.ygrid.yahoo.com:50511	0106	0107

Regions by Region Server

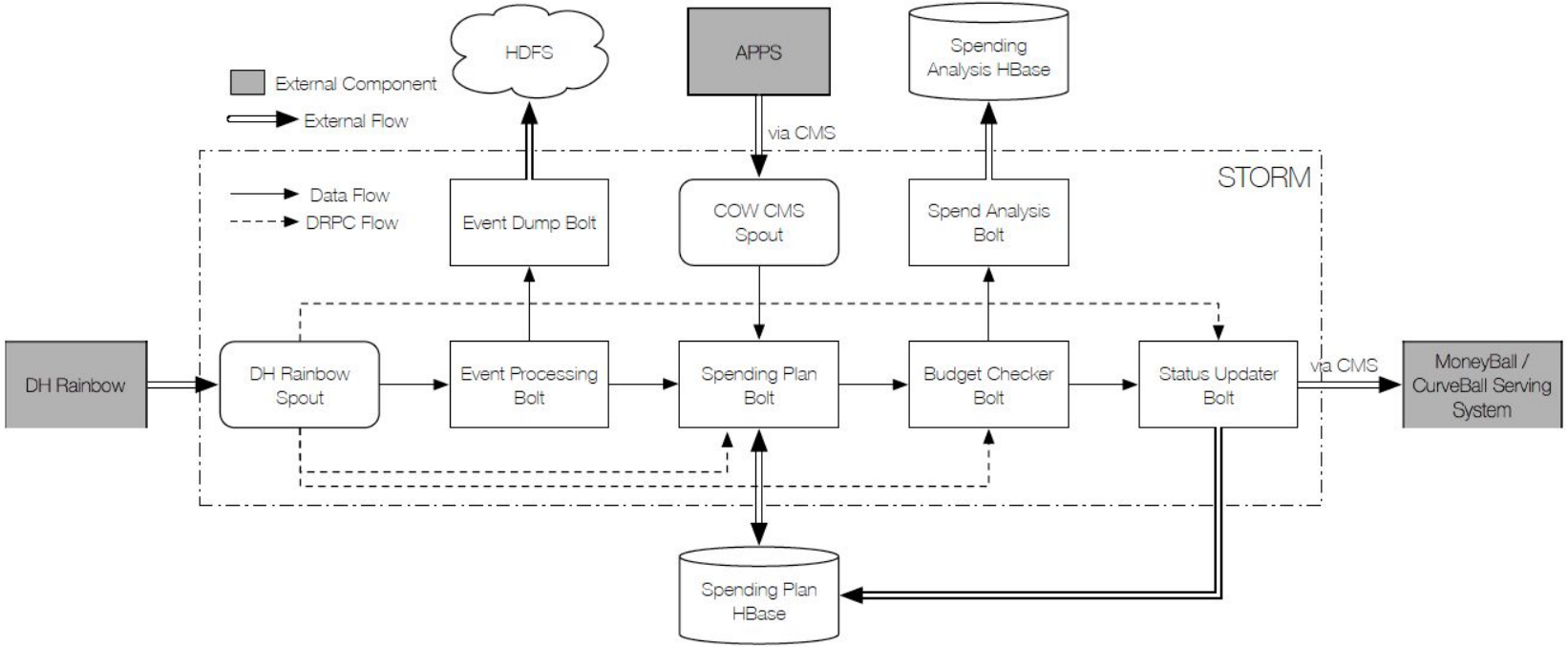
Region Server	Region Count
gsrd557n23.red.ygrid.yahoo.com:50511	97
gsrd557n24.red.ygrid.yahoo.com:50511	97
gsrd557n25.red.ygrid.yahoo.com:50511	97
gsrd557n26.red.ygrid.yahoo.com:50511	97
gsrd561n23.red.ygrid.yahoo.com:50511	97
gsrd561n24.red.ygrid.yahoo.com:50511	97
gsrd561n25.red.ygrid.yahoo.com:50511	97
gsrd561n26.red.ygrid.yahoo.com:50511	97

Accessing HBASE spend plans

- Couple of services available to access hbase spend plans easily
  - Queries budget:spend_plans table for a given object id and plan timestamp range
  - http://mw-bk-report001.bud.cb.bf1.yahoo.com:4080/query?action=splan&start=201607280000&end=201607280100&ids=344593152&objtype=256
    - https://git.corp.yahoo.com/budget/curveball_ffl_tools/blob/master/src/main/java/yahoo/budget/tools/DataLoader.java
    - https://git.corp.yahoo.com/budget/curveball_ffl_tools/blob/master/src/main/java/yahoo/budget/tools/audit/HBaseLoader.java

Budget Topology

https://docs.google.com/presentation/d/1oq2Mi0HheqllgLkCuk7EjYZS4bk_xGcc3VCtB2Xupgo/edit#slide=id.g154ae15596_0_21



Storm Topology

- Budget Fast Feedback Loop Topology
  - https://git.corp.yahoo.com/budget/curveball/blob/master/devel/src/fast_feedback_loop/src/main/java/com/yahoo/curveball/budget/ffl/topology/FFLTopology.java

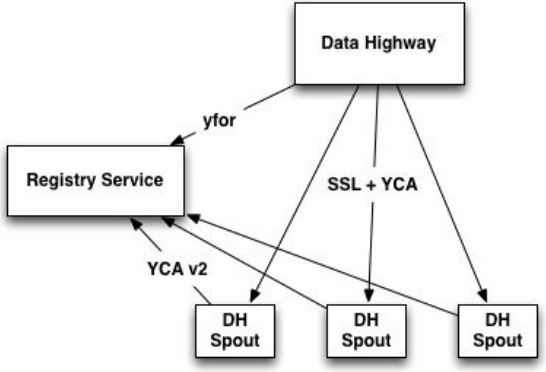
Spouts (All time)

Id	Executors	Tasks
COWCMSConsumerSpout	12	12
DHEventSpout	60	60
DRPCSpout	1	1

Bolts (All time)

Id	Executors	Tasks
BudgetCheckerBolt	40	40
BudgetStatusUpdaterBolt	20	20
EventProcessBolt	30	30
JoinResult	1	1
prepare-request	1	1
ResultCollector	1	1
ReturnResults	1	1
SpendAnalysisBolt	10	10
SpendingPlanBolt	20	20

DHEventSpout

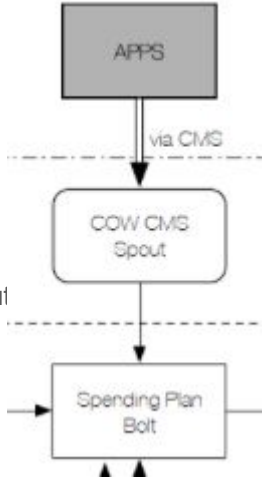


- DH uses push model to push events to Spout
  - It is different from typical pull model
  - Needs a way find to find the HTTP servers (spouts) to receive data
  - RegisteryService bridges DH and DH spouts
    - https://git.corp.yahoo.com/storm/storm-contrib/tree/master/http_spout/src/main/java/com/yahoo/spout/http
    - https://git.corp.yahoo.com/storm/storm-contrib/blob/master/http_spout/src/main/java/com/yahoo/spout/http/HttpSpoutCompat.java
- DHEventSpout
  - https://git.corp.yahoo.com/budget/curveball/blob/master/devel/src/fast_feedback_loop/src/main/java/com/yahoo/curveball/budget/ffl/operators/DHEventSpout.java
  - Receives events from DH (sm/mb events - serve/impression/clicks [actions/conversions])
  - Extracts track from avro record of DH Event and pushes to EventProcessBolt
- Registry Service
  - All spouts register HTTP server host and port under one service id - **curveball-budget-prod.red.ygrid.local**
  - curl http://registry-a.red.ygrid.yahoo.com:4080/registry/v1/virtualHost/**curveball-budget-prod.red.ygrid.local**/server

Id	Topology	Executors	Tasks
DHEventSpout	CurveballFFL	60	60

Window	Emitted	Transferred
10m 0s	83026862	83033048

COWCMSConsumerSpout



- Consumes domain object changes from CMS topic
  - SpoutDeclarer cowConsumerSpout = builder.setSpout("COWCMSConsumerSpout", new COWCMSConsumerSpout
- Parallelism
  - 12 Instances of CowConsumerSpout are running
  - There are 48 topics (AdDataDomainObjectStream[1-48])
  - Each instance is having 4 threads (48/12) and subscribing to one of the topics
  - For example task_instance_12 subscribes to AdDataDomainObjectStream(12,24,36,48)
    - https://git.corp.yahoo.com/budget/curveball/blob/master/devel/src/fast_feedback_loop/src/main/java/com/yahoo/curveball/budget/ffl/utils/CMSConsumer.java
  - Interested in Advertiser, Campaign and AdvertiserFunds and filters all other domain objects (only ON status)
    - https://git.corp.yahoo.com/budget/curveball/blob/master/devel/src/fast_feedback_loop/src/main/java/com/yahoo/curveball/budget/ffl/utils/JSONDomainObjectReader.java
- Emits tuple with (object_id+object_type, domain_object) - object_type can be either advertiser or campaign

Id	Topology	Executors	Tasks
COWCMSConsumerSpout	CurveballFFL	12	12

Window	Emitted
10m 0s	16220
3h 0m 0s	36922
1d 0h 0m 0s	1083082

EventProcessBolt

```
public class SpendEvent implements Serializable {
    long objectId;
    String eventType;
    int objectType;
    Long receiveTime;
    Long joinTime;
    Double actualSpend = 0.0;
    Double spendInUsd = 0.0;
    Integer pricingType;
    Integer advCostCurrencyId;
    Track track;

    // add transient since no need to
    // transfer these fields to downstream bolt
    transient String sectionId;
    transient int deviceId;
    transient String advertiserId;
    transient long impression;
    transient long serve;
    transient long click;
    transient long conversion;
}
```

- Event Processing
  - Receives serve, impressions, clicks and conversions
  - Extract section, device_type, receive_ts from supply section
  - Contains multiple offers (Ads), for each offer extract
    - Advertiser Id, Campaign Id, Pricing Type, Adv Cost, AdListing Type
    - Create SpendEvent (advertiser/campaign) level for each offer
      - https://git.corp.yahoo.com/budget/curveball/blob/master/devel/src/fast_feedback_loop/src/main/java/com/yahoo/curveball/budget/ffl/beans/SpendEvent.java
    - Finally emit tuple containing (object_id+object_type, SpendEvent) - object type [Advertiser/Campaign]
    - One per advertiser and one per campaign
- Caches aggregates (imps/serves/clicks/spend) for multiple dimensions for the last 2 hours in 5 min intervals
  - Aggregate imps/serves/clicks/spend for the following dimensions
    - [object_id + advertiser_id + section_id + device_id + pricing_type + track (mb/search/bing)]
  - These aggregates can be accessed through DRPC command, Here is example
    - Fetch bing spend for the last 5 min
    - http://gsbl866n01.blue.ygrid.yahoo.com:4080/drpc/query/**getEventInfo:startTsInMinute=201607222250&endTsInMinute=201607222255&comp=EventProcessBolt&track=bing&sum=spend**

EventProcessBolt - DRPC Support

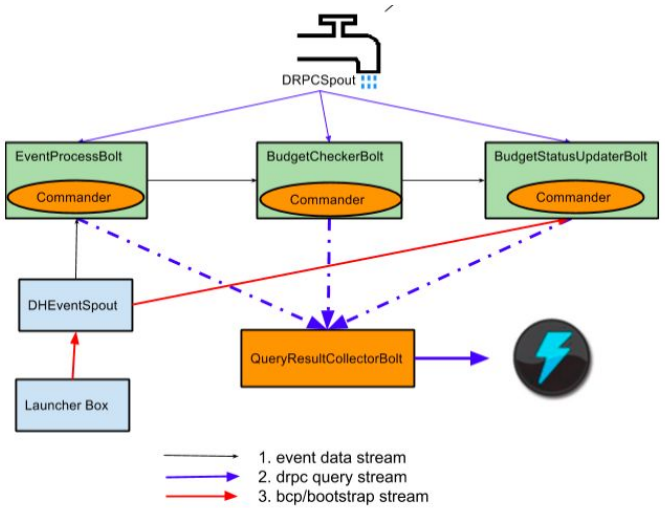
- Cached aggregates (imps/serves/clicks/spend) can be accessed at various dimensions (advertiser, campaign, section ...) through DRPC requests
- If the input tuple contains “request” (DRPC request), args for the request are extracted and delegated to EventProcessDRPC via Command Object
  - https://git.corp.yahoo.com/budget/curveball/blob/master/devel/src/fast_feedback_loop/src/main/java/com/yahoo/curveball/budget/ffl/operators/drpc/EventProcessDRPC.java
  - EventProcessDRPC can filter cached aggregates by (start/end ts, track, advertiserId, campaign id)
  - Command supports various POST operations on the result set from EventProcessDRPC
    - Filter, Group BY, Order BY, SELECT, Limit, SUM
      - https://git.corp.yahoo.com/budget/curveball/blob/master/devel/src/fast_feedback_loop/src/main/java/com/yahoo/curveball/budget/ffl/operators/drpc/Commander.java

Id	Topology	Executors	Tasks
EventProcessBolt	CurveballFFL	30	30

Window	Emitted	Transferred	Execute latency (ms)	Executed
10m 0s	2473415	2473602	0.028	44639078
3h 0m 0s	42619540	42619540	0.030	860292219
1d 0h 0m 0s	463393326	463393327	0.029	8571182501

DRPC Commands

- DRPC calls can be used to query results from storm topologies
- Example: Get last 5M bing spend
  - http://gsbl866n01.blue.ygrid.yahoo.com:4080/drpc/query/**getEventInfo:startTsInMinute=201607222250&endTsInMinute=201607222255&comp=EventProcessBolt&track=bing&sum=spend**
    - **{"result":[{"spend":18504.36}],"upstream":30,"time":1469228940}**
  - Please refer to DRPC Architecture for more details
- Check EventProcessBolt from Storm UI
  - http://iridiumred-ni.red.ygrid.yahoo.com:9999/component.html?id=EventProcessBolt&topology_id=CurveballFFL-108-1469123775



QueryResultCollectorBolt

- Collects DRPC command results from participating Bolts for a given request
- Forwards combined result to JoinResult Bolt

```
BoltDeclarer resultCollector = builder.setBolt("ResultCollector", new QueryResultCollectorBolt(conf, PREPARE_ID))
        .fieldsGrouping("BudgetStatusUpdaterBolt", "result", new Fields("request"))
        .fieldsGrouping("BudgetCheckerBolt", "result", new Fields("request"))
        .fieldsGrouping("EventProcessBolt", "result", new Fields("request"))
        .fieldsGrouping("SpendingPlanBolt", "result", new Fields("request"))
        .fieldsGrouping(PREPARE_ID, PrepareRequest.RETURN_STREAM, new Fields("request"));
```

- Creates a thread for every request it receives from PrepareRequest Bolt
  - Blocks for the results
  - When result arrives for a given request from one of the bolts
    - Waiting thread is interrupted and records result for that particular request
    - Starts another thread waits for 1 sec to collect all results and finally sends result
  - https://git.corp.yahoo.com/budget/curveball/blob/master/devel/src/fast_feedback_loop/src/main/java/com/yahoo/curveball/budget/ffl/operators/QueryResultCollectorBolt.java

DRPC Commands - EventProcessBolt

- Aggregates imps/serves/clicks/spend for the following dimensions in 5min intervals (last 2 hours)
  - [object_id + advertiser_id + section_id + device_id + pricing_type + track (mb/sm/bing)]
- Can query sum (spend, impression, serve, click) by track (mb/sm/bing)
  - http://gsbl866n01.blue.ygrid.yahoo.com:4080/drpc/query/getEventInfo:startTsInMinute=201607231625&endTsInMinute=201607231630&sum=spend&sum=impression&sum=click&sum=serve&track=sm
- Get top 10 advertiser's spend and clicks for track="sm" order by spend (desc)
  - http://gsbl866n01.blue.ygrid.yahoo.com:4080/drpc/query/getEventInfo:startTsInMinute=201607231625&endTsInMinute=201607231630&track=sm&groupby=advertiser&merge=spend&merge=click&orderby=spend&desc=true&limit=10
- Filter & SELECT example
  - Add a couple of examples

```
int spendingPlanBoltParallelism = conf.get(FFLConstants.STORM_SPENDING_PLAN_BOLT_PARALLELISM, 10);
BoltDeclarer spendingPlanBolt = builder
        .setBolt("SpendingPlanBolt", new SpendingPlanBolt(conf), spendingPlanBoltParallelism)
        .customGrouping("EventProcessBolt", "data", new KeyBasedPartitionGrouping(1))
        .customGrouping("COWCMSConsumerSpout", "data", new KeyBasedPartitionGrouping(1))
        .allGrouping("DHEventSpout", "tweak");
```

SpendingPlan Bolt

- SpendingBolt Inputs
  - Receives SpendEvents from EventProcessBolt
  - Receives Advertiser/Campaign/AdvertiserFunds domain objects from COWCMSConsumerSpout
- Maintains budget cache
  - Latest spend plan information for a given object (advertiser/campaign) - next slide
  - Latest spend plan is loaded from HBASE (budget:spending_plans)
- Output Tuple
  - `declarer.declareStream("data", new Fields("key", "flag", "spendingPlan", "spendEvent"));`
- Processing
  - SpendEvent - Retrieve current spend plan info for SpendEvent object (advertiser/campaign)
  - DomainObject - If there is no spend plan for this object in current cache, generate one
    - Handles new advertisers and campaigns

Id	Topology	Executors	Tasks
SpendingPlanBolt	CurveballFFL	20	20

BudgetCache

- Maintains [Budget, Status] for a given advertiser/campaign object
  - Key is [id + “_” + type of object (advertiser/campaign)]
- Initially cache is loaded from HBASE budget:spending_plans table
  - First get latest spend plan timestamp from meta table
    - hbase(main):003:0> get 'budget:spending_plans_meta’, '3_latest_spend_plan_start_ts'
      b:ts timestamp=1469313616375,value={"interval_start":1469311800,"plan_gen_start":1469313000}
  - Load that particular spend plan from budget:spending_plans table
    - Example Key: “2301|3 |1469313900|1000013 |257”
    - day+hash(id+”_”+type)%25 | version | plan timestamp | id | type(advertiser/campaign)
    - Scan each partition with start key 23[00-25]|<plan ts>| (25 partitions for a given day)
- Cache Refresher
  - Thread is dedicated to refresh cache by periodically checking latest spend plan timestamp from meta table
- Each spending plan record also contains spend plan managed info
  - If advertiser is managed, managed_id contains reseller id
- Construct resellers list and <advertiser_id, reseller_id> mapping

```
public class Status {
    private float trate;
    private final long sn;
    private ThrottleType throttleType;
```

```
public class Budget {

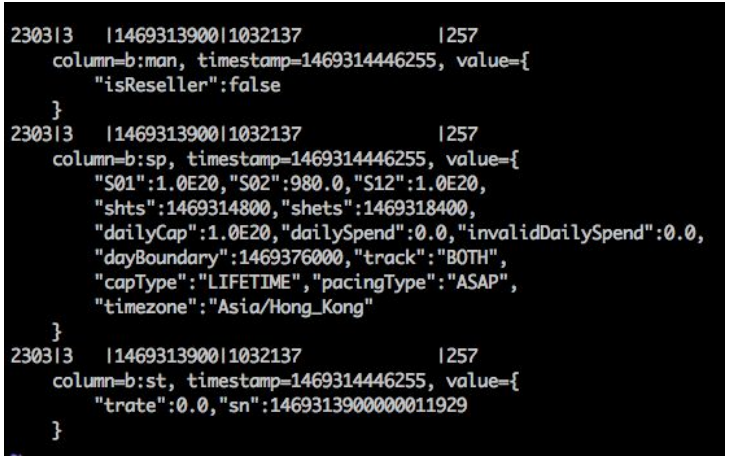
    private double S01;
    private double S02;
    private double S12;
    // second hour timestamp for negative spend p
    private long shts;
    // second hour end timestamp
    private long shets;
    private double dailyCap;
    private double dailySpend;
    private double invalidDailySpend;
    private long dayBoundary;
    // whether the id is covered by smart pacing
    private PacingAlgo smartAlgo;
    private Track track;
    private CapTypeEnum capType;
    private PacingTypeEnum pacingType;

    // this is for temp spending plan, default va
    // while in sos, this is treated as original
    private Long firstIntervalStart;
    // this is for sos to record the new sp's int
    private Long intervalStart;
    private String pacingBktId = null;
    private String timezone = null;

    // Campaign level's distKey. SPAdj tries to f
    // based on distKey (if there is) to generate
    private String distKey;
```

Spending Plan Records

- hbase(main):001:0> get 'budget:spending_plans', '2303|3 |1469313900|1032137 |257'
  - Managed Info
    - If reseller, isReseller set to TRUE
    - If advertiser is managed, then it contains managed advertiser id (reseller)
  - SpendPlan Info
  - Throttling Rate Info
- Budget Distributions - budget:spending_plans_meta
  - Budget pacing_type can be weighted by hour.
  - Campaign can have a budget distribution key
  - Each budget distribution key has 24 weights (one for each hour of the day) - sum is 1
    - [{"weights":[{"index":0,"weight":0.1},{"index":1,"weight":0.2}],"distKey":"distkey1"},{"weights":[{"index":0,"weight":0.2},{"index":1,"weight":0.1}],"distKey":"distkey2"}]



SpendingPlan Partition

```
int spendingPlanBoltParallelism = conf.get(FFLConstants.STORM_SPENDING_PLAN_BOLT_PARALLELISM, 10);
BoltDeclarer spendingPlanBolt = builder
        .setBolt("SpendingPlanBolt", new SpendingPlanBolt(conf), spendingPlanBoltParallelism)
        .customGrouping("EventProcessBolt", "data", new KeyBasedPartitionGrouping(1))
        .customGrouping("COWCMSConsumerSpout", "data", new KeyBasedPartitionGrouping(1))
        .allGrouping("DHEventSpout", "tweak");
```

- SpendingPlanBolt Partitioning Scheme
  - Uses KeyBasedPartitionGrouping to receive events from EventProcessBolt/COWCMSConsumerSpout
  - https://git.corp.yahoo.com/budget/curveball/blob/master/devel/src/fast_feedback_loop/src/main/java/com/yahoo/curveball/budget/ffl/operators/KeyBasedPartitionGrouping.java
  - SpendEvent/DomainObject - PartitionKey is “object id” + Type (Type can be either advertiser or campaign)
    - Receiving SpendingBolt Task id = hash(object_id+type) % total number of SpendingBolt Tasks
- BudgetCache
  - Spend Plan contains spend plan for each advertiser and campaign (approx 12M entries)
  - Each SpendingPlan Task loads only subset of them
    - Task id = hash(object_id+type) % total number of SpendingBolt Tasks
    - https://git.corp.yahoo.com/budget/curveball/blob/master/devel/src/fast_feedback_loop/src/main/java/com/yahoo/curveball/budget/ffl/utils/HBaseCacheLookup.java
  - Resellers are loaded in every SpendingBolt

```
int groups = callbackObject.getNumTasks(true);
int index = callbackObject.getCurTaskIndex(true);
if ((Math.abs(partition.hashCode()) % groups) == index) {
    return true;
}
```

```
int budgetCheckerBoltParallelism = conf.get(FFLConstants.STORM_BUDGET_CHECKER_BOLT_PARALLELISM, 10);
int boltRedundantFactor = conf.get(FFLConstants.BUDGET_CHECKER_BOLT_REDUNDANT_FACTOR, 1);
BoltDeclarer budgetCheckerBolt = builder
        .setBolt("BudgetCheckerBolt", new BudgetCheckerBolt(conf), budgetCheckerBoltParallelism)
        .customGrouping("SpendingPlanBolt", "data", new KeyBasedPartitionGrouping(boltRedundantFactor))
        .allGrouping("DHEventSpout", "tweak").allGrouping("SpendingPlanBolt", "meta");
```

BudgetCheckerBolt

- Computes throttling rate for a given advertiser/campaign
  - Based on current spend plan and accumulated spend
- Input Stream
  - Data Stream - ((object_id+type), Type of Spend Plan, SpendingPlan, SpendEvent)
  - Partitioned by (object_id+type)
- Output Stream
  - `declarer.declareStream("data", new Fields("key", "throttleRate", "index", "uptime"))`
- Redundancy
  - same object_id+type is sent to multiple backup tasks
  - https://git.corp.yahoo.com/budget/curveball/blob/master/devel/src/fast_feedback_loop/src/main/java/com/yahoo/curveball/budget/ffl/operators/KeyBasedPartitionGrouping.java

```
if (!StringUtils.isEmpty(partitionKey)) {
    int hashcode = Math.abs(partitionKey.hashCode());
    int index = hashcode % backupNumTasks;
    for (int i = 0; index + i * backupNumTasks < totalNumTasks; i++) {
        taskIds.add(tagetTasks.get(index + i * backupNumTasks));
    }
}
```

Id	Topology	Executors	Tasks
BudgetCheckerBolt	CurveballFFL	40	40

CachedSpendInfo - BudgetCh

```
public class CachedSpendInfo {
    private ConcurrentHashMap<String, ConcurrentSkipListMap<Long, SpendInfo>> spendMap;
    private TimeSliceQueue<Map<String, SpendInfo>> rollupSpendQueue;
    private ConcurrentHashMap<String, Pair<Long, Float>> rateMap;
    private ConcurrentHashMap<String, Long> statusMap;
    private ConcurrentHashMap<String, Track> trackMap;
}
```

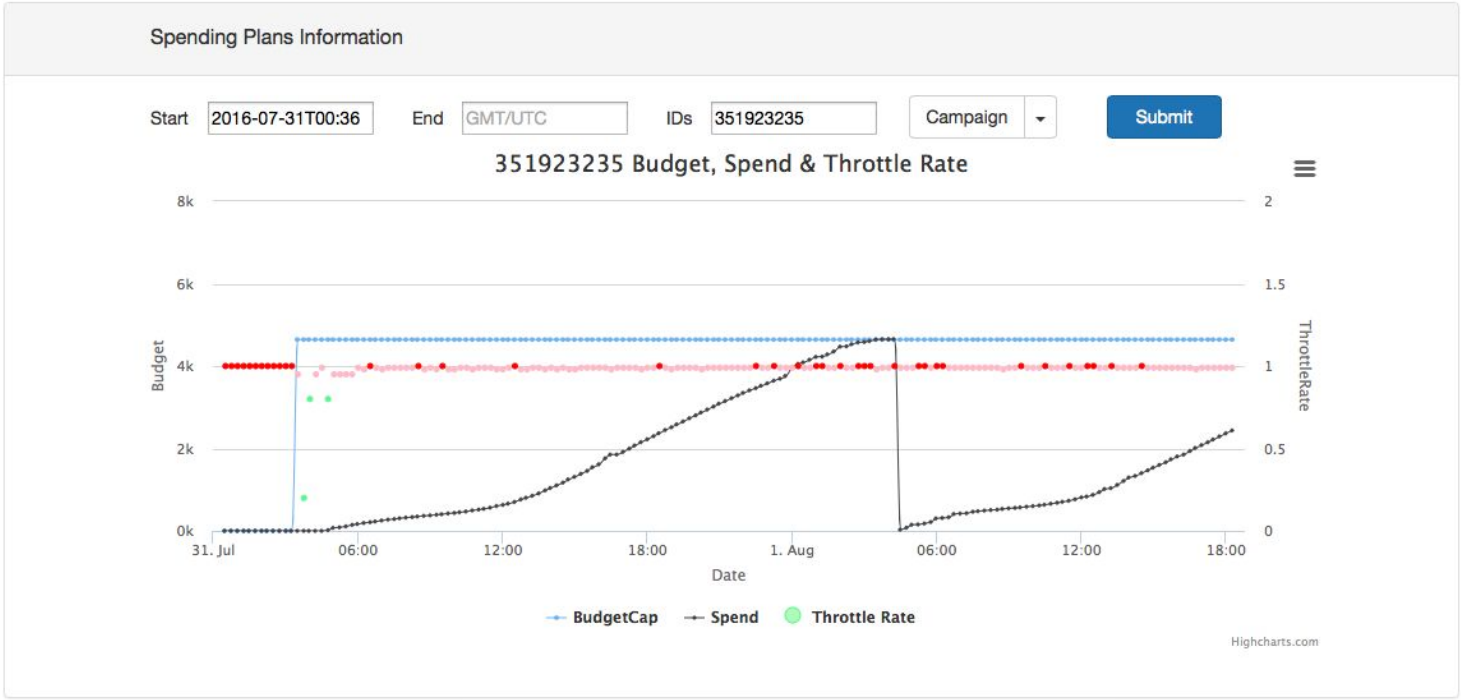
- CachedSpendInfo captures real-time spend info (moving window)
  - spendMap contains minute level spend info for a given object id (advertiser/campaign)+ type
  - rollupSpendQueue contains 5 min level aggregates for a given object_id + type
    - https://git.corp.yahoo.com/budget/curveball/blob/master/devel/src/fast_feedback_loop/src/main/java/com/yahoo/curveball/budget/ffl/beans/CachedSpendInfo.java
- Processing Incoming Tuple
  - Tuple contains - ((object_id+type), Type of Spend Plan, SpendingPlan, SpendEvent)
  - For normal spend plan,
    - Update spendMap cache to accumulate spend info
    - Check overspend (compare s01, s02 and s12 limits with accumulated spend)
    - If there is overspend, then emit throttle rate
      - Tuple ((object_id + type), throttle rate, task index, bolt uptime)
  - Type of spend plan indicates what type of operation to be performed
    - Example: Type of Spend Plan can be “DUMMY” indicates spend plan got refreshed
      - It may require to check overspend and generate throttle rate
    - For other types of spend plans
      - https://docs.google.com/document/d/1RPN0zHfXL1ct6kMqjtoTLc01vRJ2tbxKT5NxsFTlmWg/edit?pli=1

Spend Pacing Algorithms

- Coming Soon :-)
  - Yet to review (next couple of days)

Smart Pacing - Example

- http://yo/budget (spend plans - hbase: budget:spend_plans table)



DRPC Commands - BudgetCheckerBolt

```
public class DrpcSpendInfo {
    double localSpends;
    double usdSpends;
    double usdSpendInUsd;
    double nonUsdSpendInUsd;
    double usdRemainingCap;
}
```

- Get spend info (time interval)
  - https://git.corp.yahoo.com/budget/curveball/blob/master/devel/src/fast_feedback_loop/src/main/java/com/yahoo/curveball/budget/ffl/operators/drpc/BudgetCheckerDRPC.java
  - Maintains 5 min level aggregate SpendInfo at track (mb/sm/bing) for a given object_id + type
    - Object_id can be either advertiser/campaign
    - `TimeSliceQueue<Map<String, Map<Track, DrpcSpendInfo>>> spendQueue;`
  - Example - Advertiser level aggregate spend - mb
    - http://gsbl866n01.blue.ygrid.yahoo.com:4080/drpc/query/getSpendInfo:startTsInMinute=201607262310&endTsInMinute=201607262315&comp=BudgetCheckerBolt&sum=spend&type=256&track=mb
- Get throttle rates Summary
  - Can query latest throttle rates for advertiser/campaign objects (getId)
    - How many campaigns are not throttled currently (on search track)
      - http://gsbl866n01.blue.ygrid.yahoo.com:4080/drpc/query/getId:sum=count&comp=BudgetCheckerBolt&type=256&trate=0&track=sm

BudgetStatusUpdaterBolt

- Receives throttle rate message for a given object (advertiser/campaign)

```
int budgetStatusUpdaterParallelism = conf.get(FFLConstants.STORM_BUDGET_STATUS_UPDATE_BOLT_PARALLELISM, 10);
BoltDeclarer budgetStatusUpdaterBolt = builder
        .setBolt("BudgetStatusUpdaterBolt", new BudgetStatusUpdaterBolt(conf, producersInfo),
                budgetStatusUpdaterParallelism).allGrouping("DHEventSpout", "tweak")
        .customGrouping("BudgetCheckerBolt", "data", new KeyBasedPartitionGrouping(1));
```

- Incoming Tuple
  - declarer.declareStream("data", new Fields("key", "throttleRate", "index", "uptime"))
  - Key - Object_id + Type (Object Id can be either advertiser or campaign id)
  - Index - Incoming task id, uptime - Uptime of the incoming bolt.
  -
- BudgetStatsUpdaterBolt - Parallelism
  - Incoming stream is partitioned by (object_id + type) - (20 instances)
  - Have redundancy in BudgetCheckerBolt
    - More than one instance of BudgetCheckerBolt is processing spend event for a given advertiser/campaign object (fault tolerant redundancy)
    - Multiple BudgetCheckerBolts (currently 2) are actually sending throttle rate messages for a given object

```
private Map<Integer, Integer> redundantCurrentSender;
private Map<Integer, long[]> redundantSenderUptime;
```

Handling Multiple Throttle Messages

- How can we handle multiple throttle messages in BudgetStatusUpdater Bolt for a given advertiser/campaign object receiving from multiple BudgetCheckerBolt instances ?
  - Maintain current sender task id for a given partition
  - Also maintain uptimes of all sender tasks for a given partition
    - https://git.corp.yahoo.com/budget/curveball/blob/master/devel/src/fast_feedback_loop/src/main/java/com/yahoo/curveball/budget/ffl/operators/BudgetStatusUpdaterBolt.java
  - Update uptime for the current task
    - Uptime => Start Time of Bolt
    - Choose message with larger Uptime

```
int index = (Integer) input.getValueByField("index");
long uptime = (Long) input.getValueByField("uptime");
int boltIndex = index % redundantBlockSize;
int blockIndex = index / redundantBlockSize;

long[] uptimes = redundantSenderUptime.get(boltIndex);
uptimes[blockIndex] = uptime;
if (blockIndex == redundantCurrentSender.get(boltIndex)) {
    addToProcessQueue(cmsRecord);
} else {
    long curSenderUptime = uptimes[redundantCurrentSender.get(boltIndex)];
    if (uptime < curSenderUptime) {
        redundantCurrentSender.put(boltIndex, blockIndex);
        addToProcessQueue(cmsRecord);
    }
}
```

CMS Throttle Message Generation

- Buffer throttle rate messages before publishing to CMS
- Each throttle message is associated with send sequence number (ssn)
  - Current Spend Plan Timestamp + sequence number
  - Each task persists current sequence number to Spend Plan meta table
    - Message generated by new spending plan must have higher sequence numbers
  - hbase(main):013:0> scan 'budget:spending_plans_meta'

```
ffl_max_sequence_num                column=b:ssn10, timestamp=1469147486397, value=1469145610000005219
ffl_max_sequence_num                column=b:ssn11, timestamp=1469147422979, value=1469145610000001135
ffl_max_sequence_num                column=b:ssn12, timestamp=1469147421138, value=1469145610000000884
ffl_max_sequence_num                column=b:ssn13, timestamp=1469147707725, value=1469145610000001183
```

  - Also updates throttle msg status of the object to HBASE spending plan table
- CMS publisher thread (every 100ms) processes all buffered messages and publish to CMS topic
- Only primary colo publishes messages to CMS
  - Current primary is stored in HBASE spend meta table

```
fl_max_sequence_num                 column=b:primary, timestamp=1469147776588, value=0
```

```
int analysisBoltParallelism = conf.get(FFLConstants.STORM_SPEND_ANALYSIS_BOLT_PARALLELISM, 1);
BoltDeclarer spendAnalysisBolt = builder.setBolt("SpendAnalysisBolt", new SpendAnalysisBolt(conf),
        analysisBoltParallelism).customGrouping("BudgetCheckerBolt", "data", new KeyBasedPartitionGrouping(1));
```

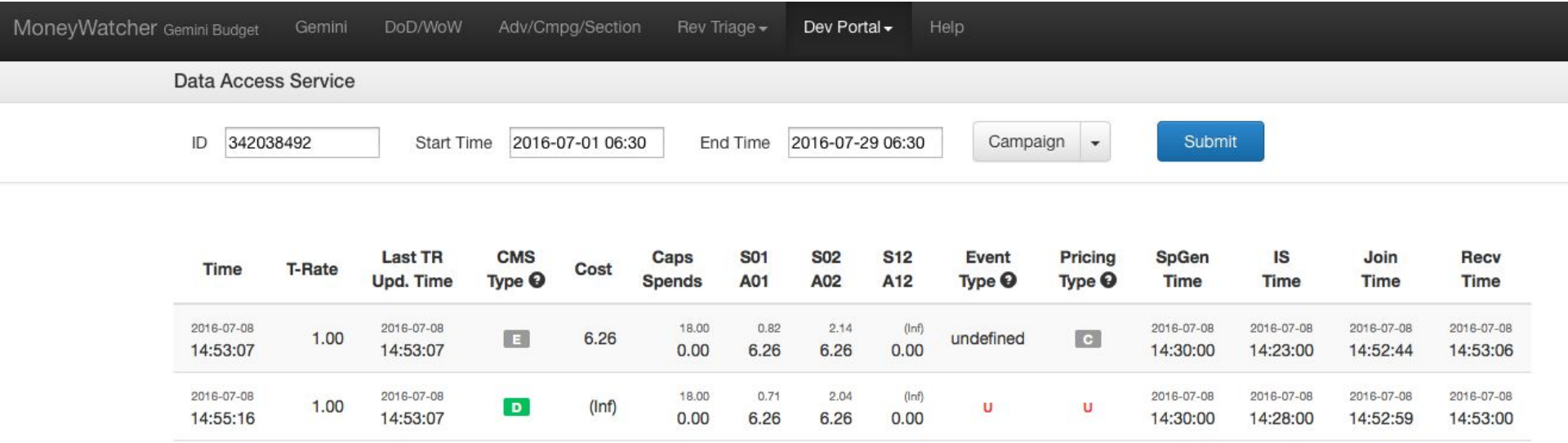
SpendAnalysisBolt

- Receives throttle messages from BudgetCheckerBolt
  - Persists them to HBASE (budget:spend_analysis) for analysis
  - Record Key - (hash(object_id+type) + object_id | type | plan TS)
    - Example Key: **\x5Cx??**342038492 |256|1467988200
  - Value Contains
    - Spend Event and Sped Plan information together
      - shts (second hour ts): July 8 15:00
      - shets (second hour end ts) : July 9 15:00)
      - Spend plan information (s01, s02, s12)
      - Actual spend Information (a01, a02, a12)
      - Overspend flag, current throttle rate, daily cap etc


```
{
  "a12":0.0,
  "curTRate":1.0,
  "lastTRateChangeTime":1467989587127,
  "eventCost":6.26,
  "s12":1.0E20,
  "isOverspend":true,
  "hbaseTRate":0.0,
  "dailySpend":0.0,
  "shts":1467990000,
  "a02":6.26,
  "a01":6.26,
  "cmsType":"Event",
  "joinTs":1467989564682,
  "lastTRate":0.0,
  "shets":1467993600,
  "dayboundary":1468036800,
  "eventType":"SMClick",
  "spGenTs":1467988200,
  "dailyCap":18.0,
  "receiveTs":1467989586927,
  "s02":2.175824175824176,
  "s01":0.8571428571428581,
  "spGenIsTs":1467987660,
  "ts":1467989587127,
  "pricingType":1
}
```

Querying Spend Analysis Table

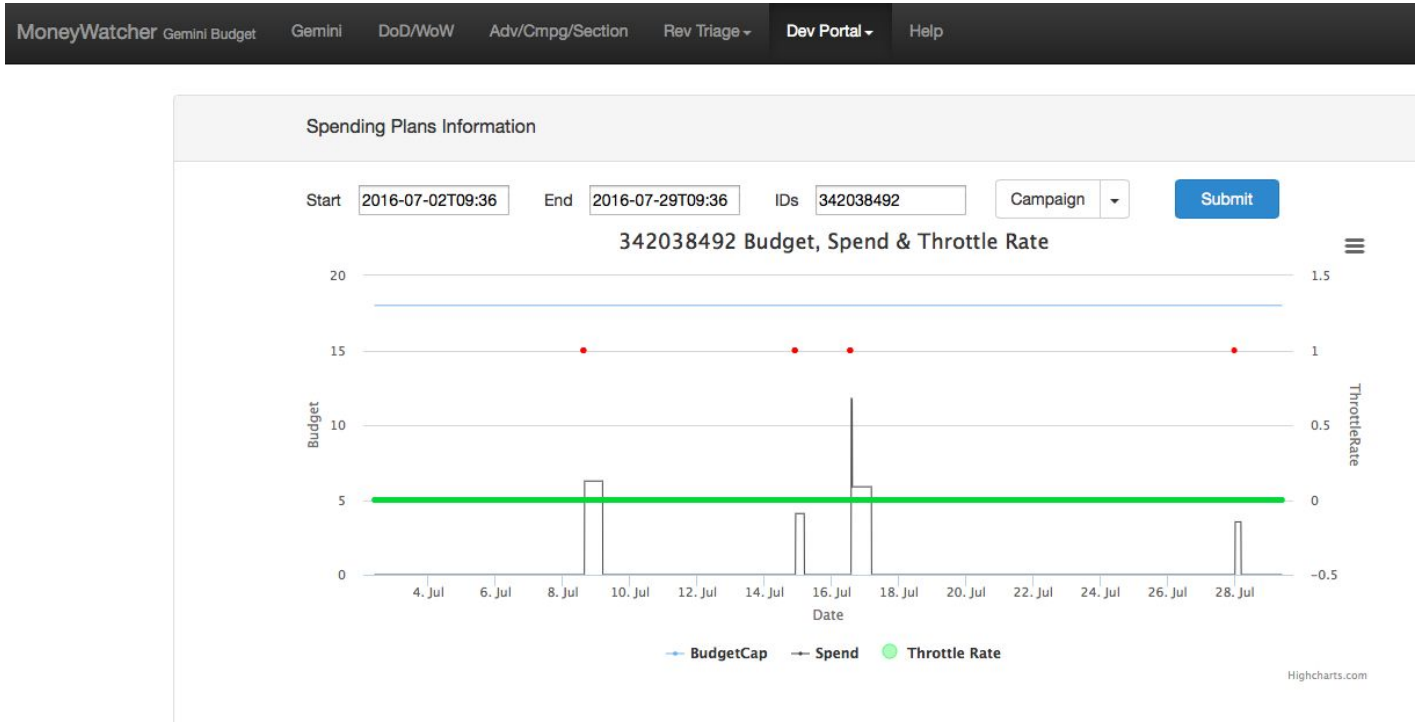
- http://yo/budget
  - Dev Portal, Data Service
  - https://git.corp.yahoo.com/budget/curveball_ffl_tools/blob/master/src/main/java/yahoo/budget/tools/DataLoader.java



Time	T-Rate	Last TR Upd. Time	CMS Type	Cost	Caps Spends	S01 A01	S02 A02	S12 A12	Event Type	Pricing Type	SpGen Time	IS Time	Join Time	Recv Time
2016-07-08 14:53:07	1.00	2016-07-08 14:53:07	E	6.26	18.00 0.00	0.82 6.26	2.14 6.26	(Inf) 0.00	undefined	C	2016-07-08 14:30:00	2016-07-08 14:23:00	2016-07-08 14:52:44	2016-07-08 14:53:06
2016-07-08 14:55:16	1.00	2016-07-08 14:53:07	D	(Inf)	18.00 0.00	0.71 6.26	2.04 6.26	(Inf) 0.00	U	U	2016-07-08 14:30:00	2016-07-08 14:28:00	2016-07-08 14:52:59	2016-07-08 14:53:00

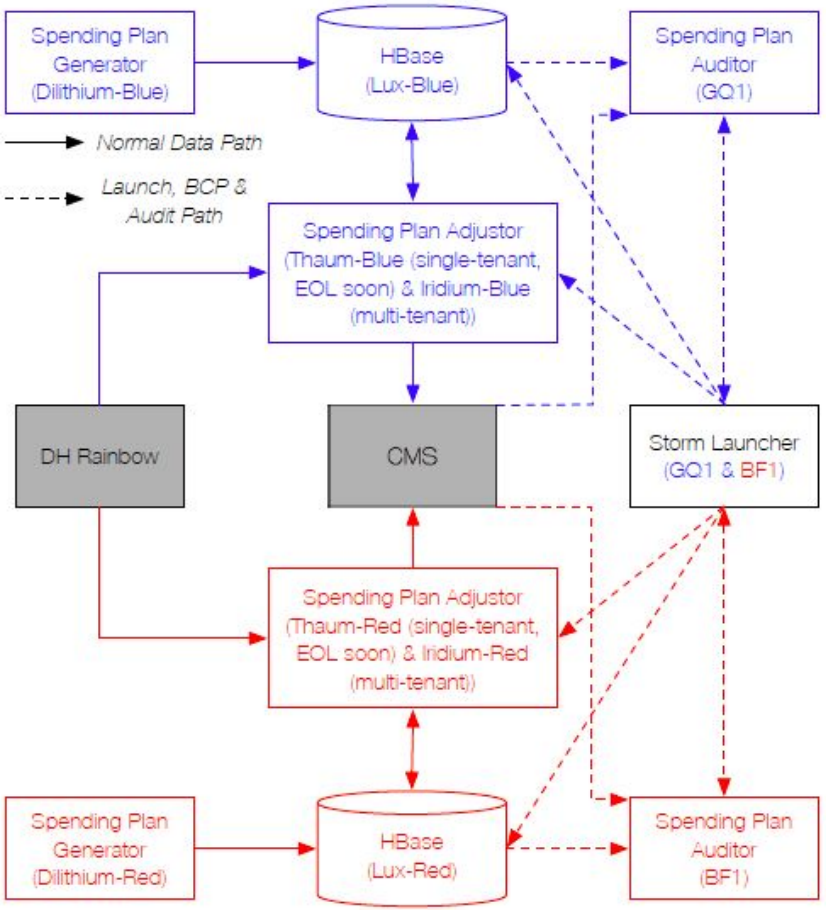
Spend Plan - Spend - Throttle rate Visualization

http://yo/budget



Deployment

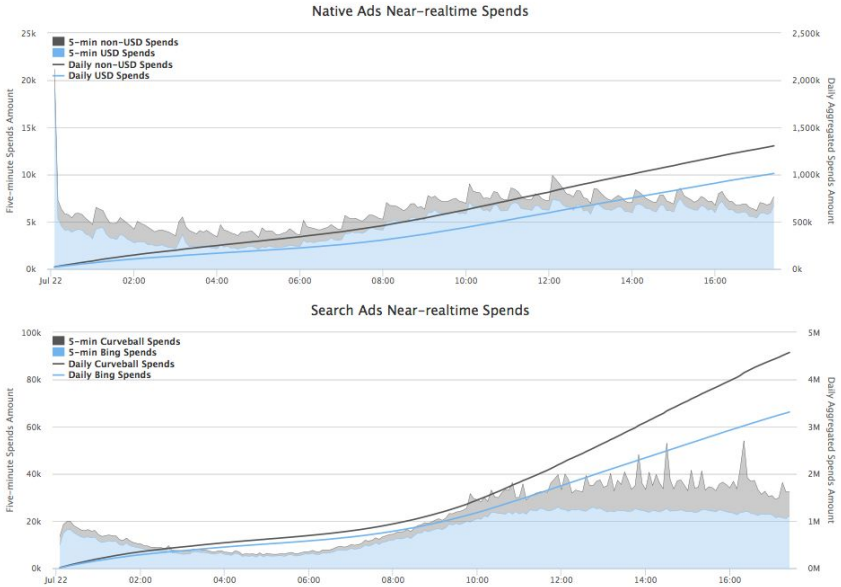
- What is Spending Plan Auditor?



http://yo/budget

- Real time dashboard showing revenue (5m intervals) in various dimensions
  - Moneyball, Gemini, Bing
  - Sections
  - Campaign level
- Gets information from mysql DB
- Mysql spend tables are getting populated every 5min running various DRPC commands
  - https://git.corp.yahoo.com/budget/curveball_operability/blob/master/drpc/backend/load.pl

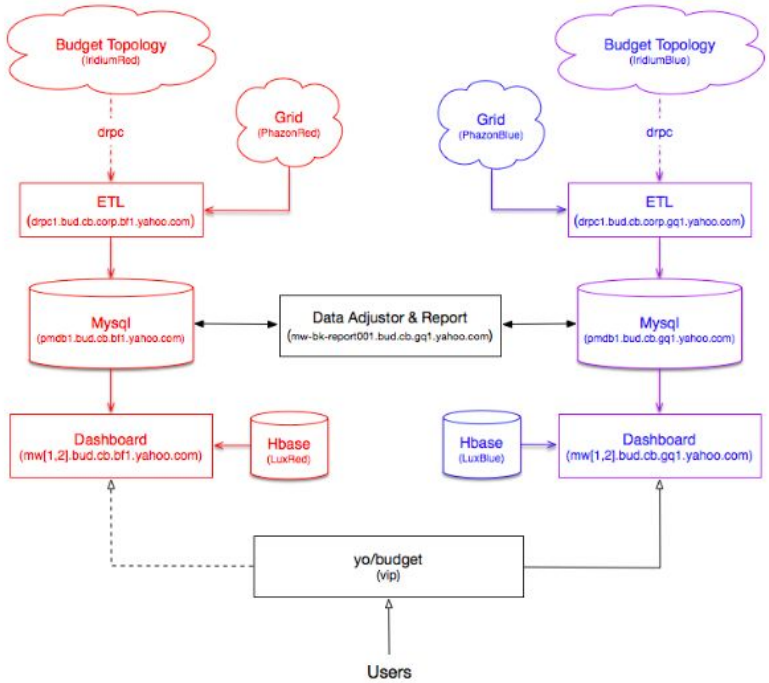
```
+------------------+
| Tables_in_pmdb   |
+------------------+
| 5m_cmp_spend     |
| 5m_section       |
| 5m_summary       |
```



http://yo/budget Architecture

- ETL
  - Extract Spend Info from STORM topology every 5 min (DRPC)
    - Drpc1.bud.cb.corp.gq1.yahoo.com
    - /home/y/bin64/drpc/load.pl
    - https://git.corp.yahoo.com/budget/curveball_operability/blob/master/drpc/backend/load.pl
  - Update mysql pmdb
    - Pmdb1.bud.cb.gq1.yahoo.com
      - 5m_cmp_spend, 5m_section, 5m_summary, revenue_triage
    - mysql> select time, round(mb_spend,0) as mb_spend, round(ss_spend,0) as ss_spend, round(bing_spend,0) as bing_spend from 5m_summary order by time desc limit 5;

time	mb_spend	ss_spend	bing_spend
2016-07-22 22:20:00	6157	8596	18946
2016-07-22 22:15:00	6382	7808	19193
2016-07-22 22:10:00	6548	7611	19190
2016-07-22 22:05:00	6639	7094	19472
2016-07-22 22:00:00	5639	10446	19926



Summary of Budget Data

- Spend Plan
  - Persisted to HDFS and HBASE (generated every 15 min)
- Spend Analysis
  - All throttle messages along with sped event info persisted to HBASE
- Spend Info
  - Real time spend is extracted from Storm (every 5 min) and persisted to mysql DB
- CMS messages
- Storm cluster Info
  - http://iridiumred-ni.red.ygrid.yahoo.com:9999/topology.html?id=CurveballFFL-25-1465410963
  - Launcher Host: launcher1.bud.cb.gq1.yahoo.com
  - DRPC Host: ssh drpc1.bud.cb.corp.gq1.yahoo.com

Bootstrap

- https://git.corp.yahoo.com/budget/curveball_bootstrap/blob/for_serving/devel/src/main/resources/launch/curveball_spending_plans_snapshot

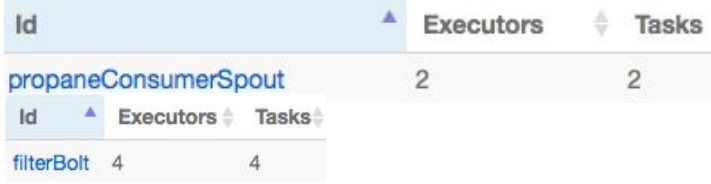
Trending Now

Propane - Audience Data

- Propane provides slices of audience data as topics like “search”, “video” etc
  - http://propane.data.yahoo.com:9999/topics
  - Example: search topic

Filter Condition
(spaceid == "980787589") and (filter_tag == null) and (event_is_user_triggered == true) and (bcookie!=null or is_logged_in==true)

- Propane spout implementation
  - https://git.corp.yahoo.com/FETL/propane-client/blob/master/src/main/java/com/yahoo/adsdata/fetl/propane/storm/PropaneSpout.java
  - Just reads propane records from specified kafka topic (Example: “search”)
    - Creates java consumer connector to the topic (`createJavaConsumerConnector`)



Id	Executors	Tasks
propaneConsumerSpout	2	2

Id	Executors	Tasks
filterBolt	4	4

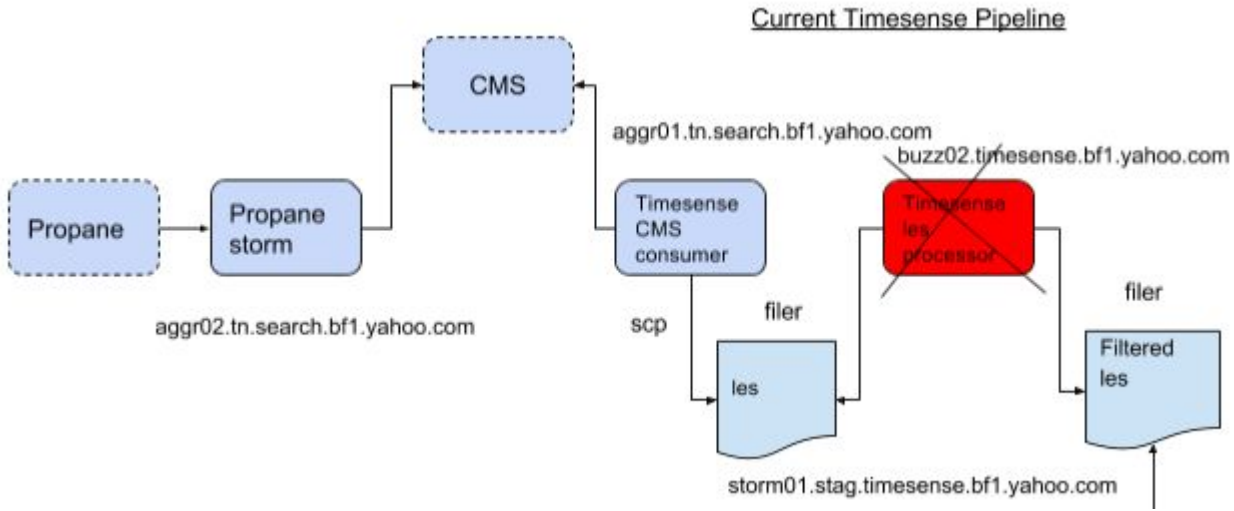
Data Ingestion - Trending Now

- Consumes search events from propane “search_results” kafka topic
  - https://git.corp.yahoo.com/trendingnow/data_ingestion/blob/master/src/main/java/timesense_dataingestion/PropaneTopology.java
  - Apply a few filters on incoming Propane records and posts transformed records to CMS
    - `[http://brokerv2.messaging.gq1.yahoo.com:4080 persistent://trendingnow/gq1/trending-prod/mt]`
- Running on a single box (not a production cluster - bad idea)
  - http://aggr02.tn.search.bf1.yahoo.com:9999/index.html
  - Check consumer.yaml for propane properties
    - Reading from “search_results” kafka topic, group_id - “PROPANE_EXAMPLES”
    - Zookeepr hosts: dp-fetlpropane-zkp-bcp4.data.ne1.yahoo.com:4080
      - ./zkCli.sh -server dp-fetlpropane-zkp-bcp4.data.ne1.yahoo.com:4080
      - get /consumers/PROPANE_EXAMPLES/offsets/search_results/ (current offsets)
      - get /kafka-manager/clusters/production_bf1/topics/search_results (topic partitions)
        - {"0":[1,30],"5":[6,37],"10":[12,43],"1":[2,31],"6":[7,38],"9":[10,42],"2":[3,34],"7":[8,39],"3":[4,35],"11":[13,44],"8":[9,40],"4":[5,36]}

```
log_config_path=/home/trendingnow/log4j.xml
subscribe_uri=persistent://trendingnow/bf1/trending-prod/test
filename_suffix=.v3.agg13.rtuf.search.bf1.yahoo.com.coke.event.stream.gz
subscribe_uri=persistent://trendingnow/bf1/trending-prod/test
filename_suffix=.v3.agg13.rtuf.search.bf1.yahoo.com.coke.event.stream.gz
dest_hostname=archiver01.timesense.bf1.yahoo.com
yca=yahoo.search_tn.arya.aggr.prod_bf1
timewindow_in_seconds=60
cms_broker=http://brokerv2.messaging.bf1.yahoo.com:4080
```

CMSConsumer

- CMSConsumer
  - Consumes from CMS topic and uploads 1 min files to filer (archiver01)
  - Aggr01.tn.search.bf1.yahoo.com
    - https://git.corp.yahoo.com/jiaqingtang/timesense_cmsconsumer/blob/master/src/main/java/cmsconsumer/CMSConsumer.java



- https://docs.google.com/document/d/1uYTb2dtdPkJxb7uy2BnEQoedNq0fuy77hTXP02Vh6UA/edit