

CDAP Metrics Perf Explained

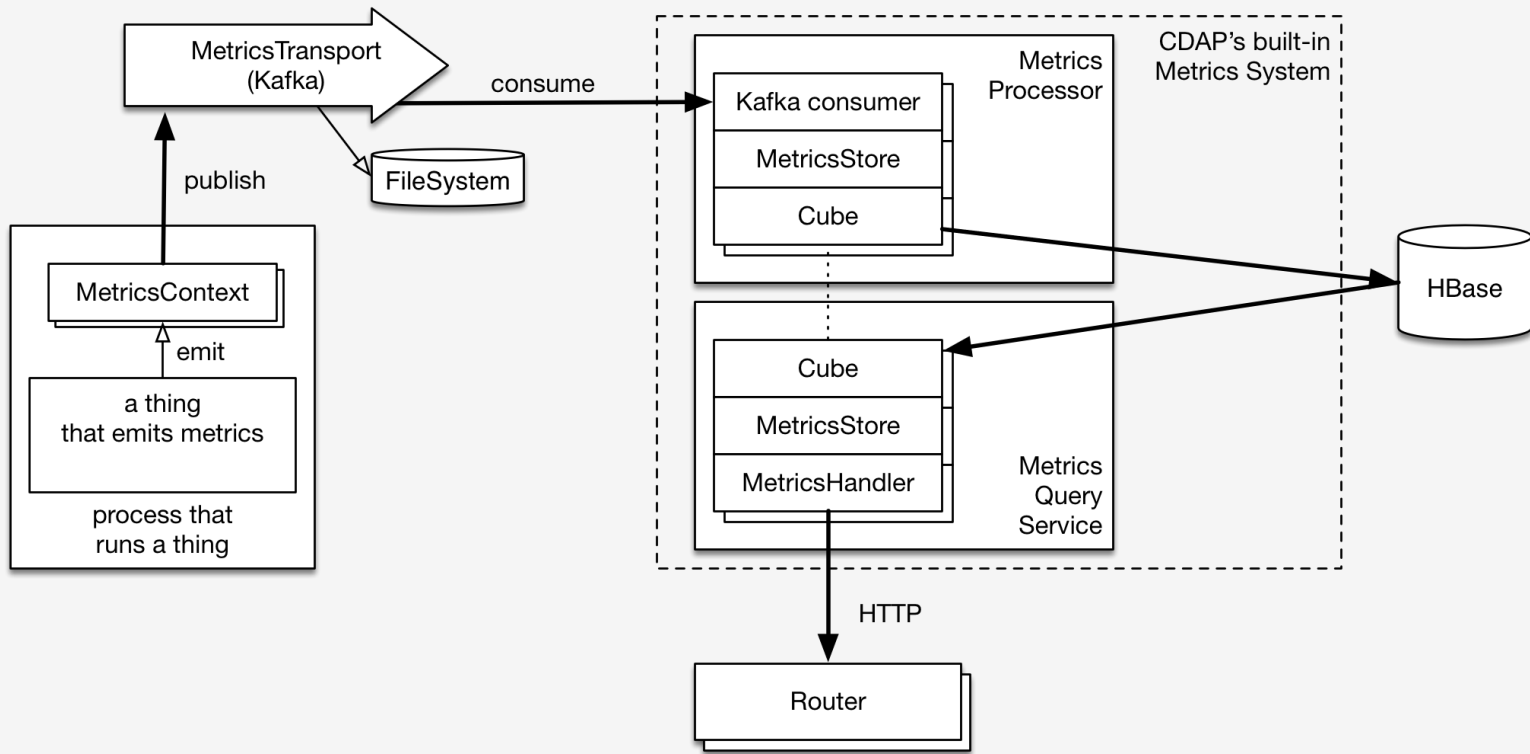
Alex

Problem

- Understanding metrics system performance is required for successful production experience of CDAP
- Customers already had problems with metrics system performance
 - Inability of processing to keep up with emitting makes operating CDAP a “flying blind” experience
 - Inability to scale out metrics processing makes impossible to get back from “flying blind” to normal operations



CDAP Metrics Overview: Data Flow



CDAP Metrics Overview: Cube

Row key format:

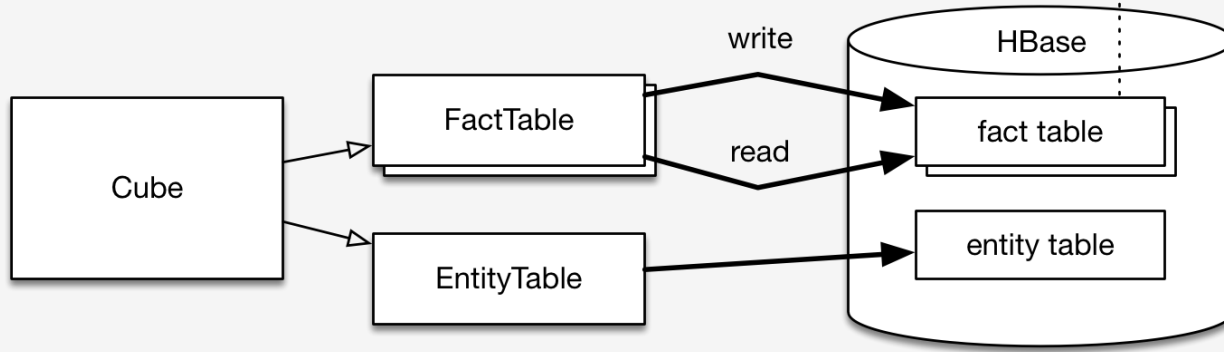
<1-byte reserved for version>

<encoded agg group><time base>

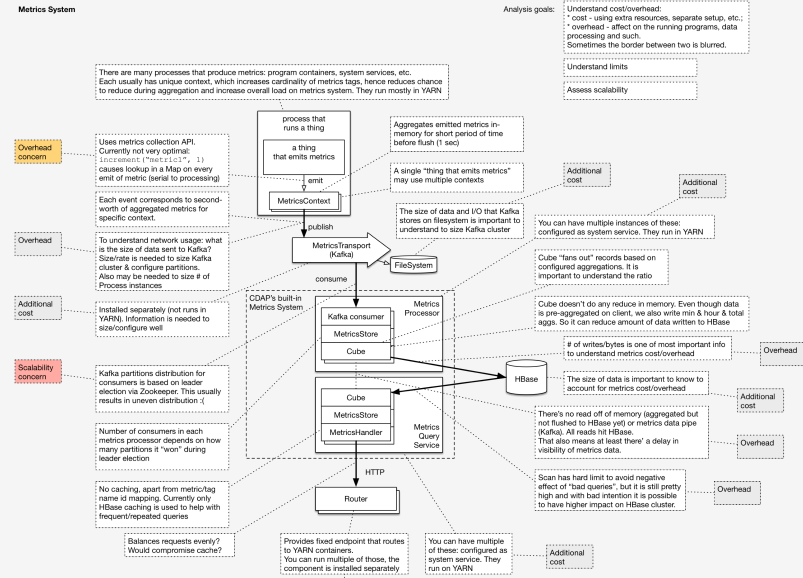
<encoded dim1 value>...<encoded dimN value>

<encoded measure name>

Note: no need to put dim name in row key, as agg group identifies dim list.

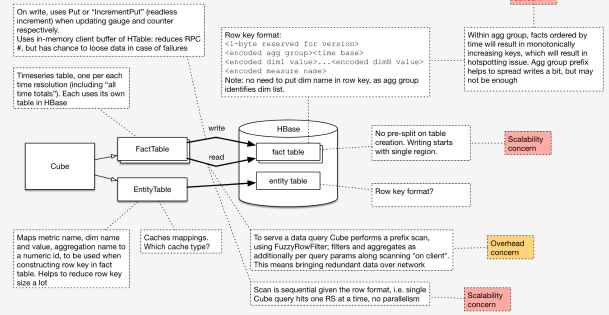


Overhead, Performance, Scalability analysis @CDAP-1127

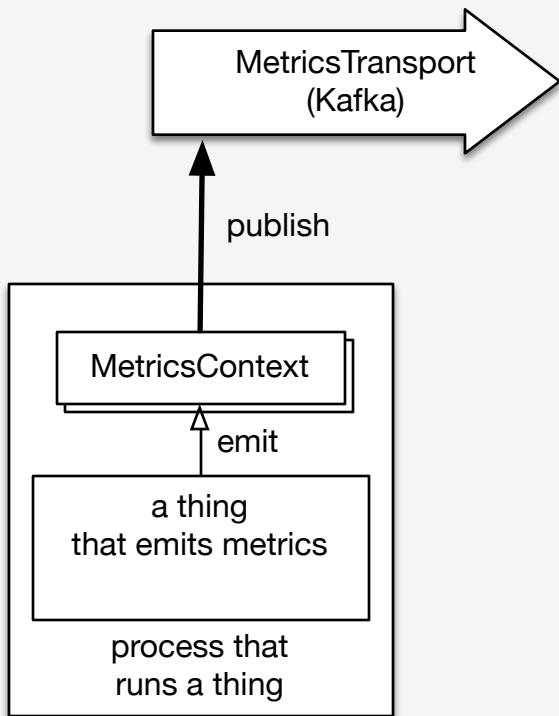


→ means "network call"

Cube (zoomed)



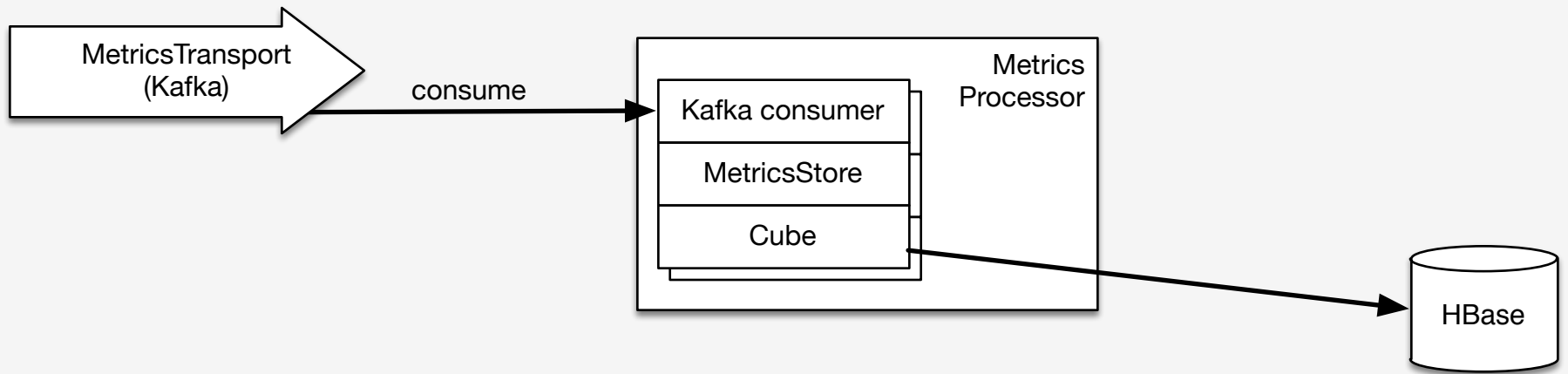
Details (continues on next slides)



A thing emits MetricValues into Kafka:

```
public class MetricValues {  
    private final long timestamp;  
    private final Map<String, String> tags;  
    private final Collection<MetricValue> metrics;  
}
```

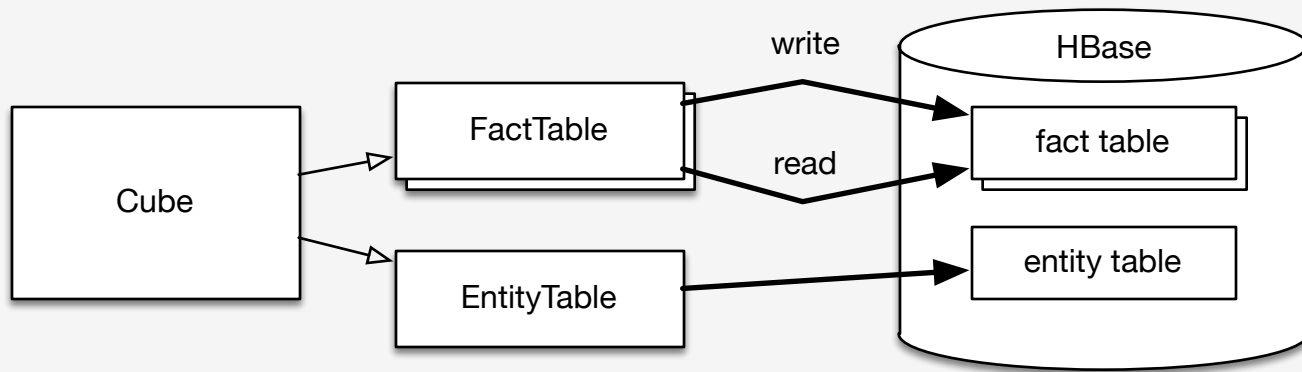
Every second, a thing emits
<# of MetricsContext> MetricValues.



All metrics data are written into single topic with multiple partitions (10 by default). There's a Kafka consumer per each partition. Consumes in micro batches (if data available). At consuming MetricValues is translated into a CubeFact:

```
public class CubeFact {  
    private final long timestamp;  
    private final Map<String, String> dimensionValues;  
    private final Collection<Measurement> measurements;  
}
```

and written into a Cube.

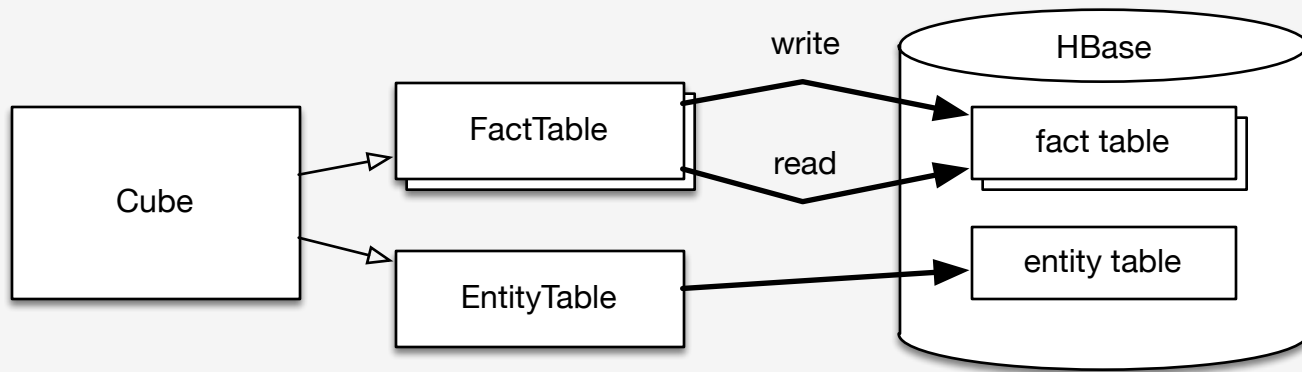


Cube is configured with what to pre-aggregate (for faster querying). E.g.:

- by_namespace
- by_namespace*by_app
- by_namespace*by_app*by_flow*by_run*by_flowlet*by_instance

On average, one CubeFact is fanned out into 3 Facts:

```
public final class Fact {  
    private final long timestamp;  
    private final List<DimensionValue> dimensionValues;  
    private final Collection<Measurement> measurements;  
}
```

Facts are written into multiple pre-aggregated time resolutions:
1sec, 1min, 1hr, “totals”.

Aggregated data for each time resolution is stored in its own
HBase table.

Row key format:

<1-byte reserved for version>

<encoded agg group><time base>

<encoded dim1 value>...<encoded dimN value>

<encoded measure name>

Note: no need to put dim name in row key, as agg group identifies dim list.

metrics contexts = # CubeFacts

Facts fanned out = # CubeFacts * # aggregations

Facts written = # Facts * # resolutions

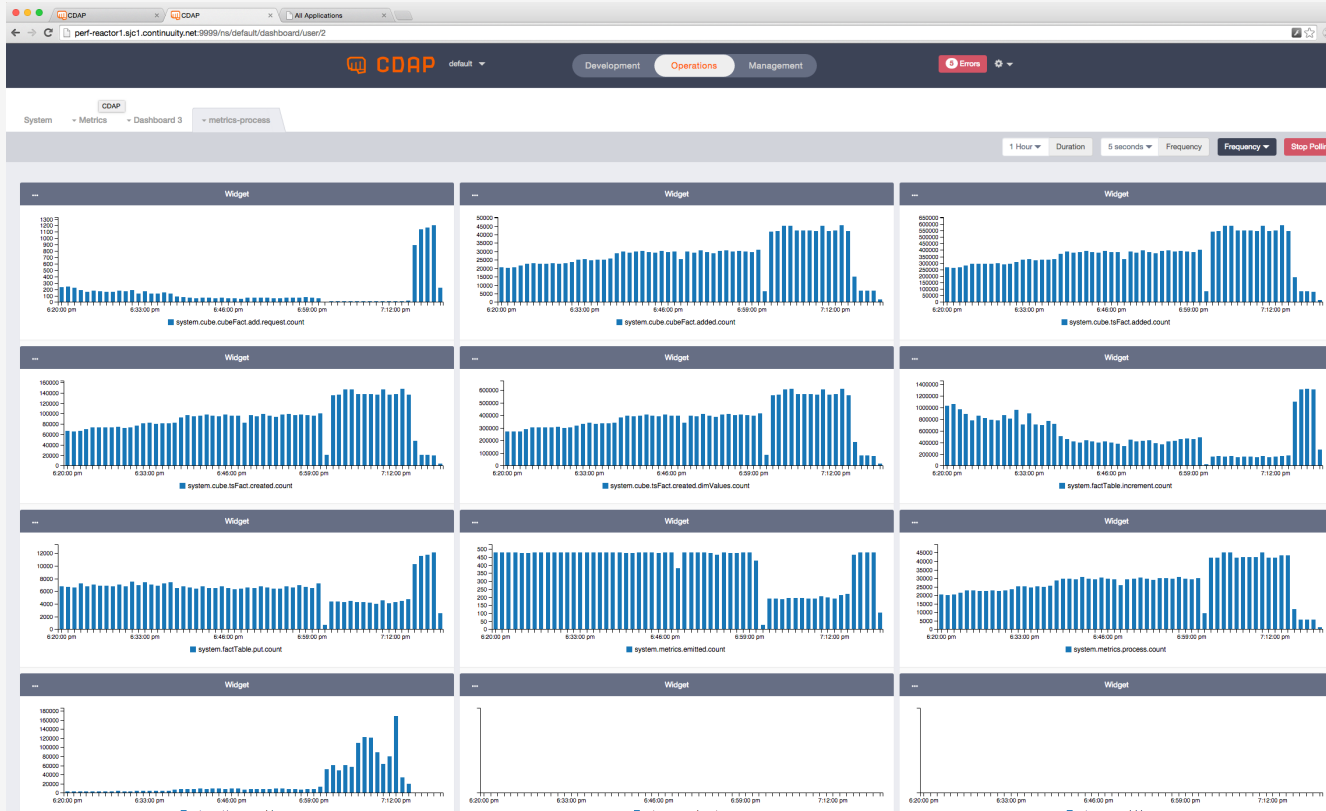
HBase records written = # of measurements in Facts written

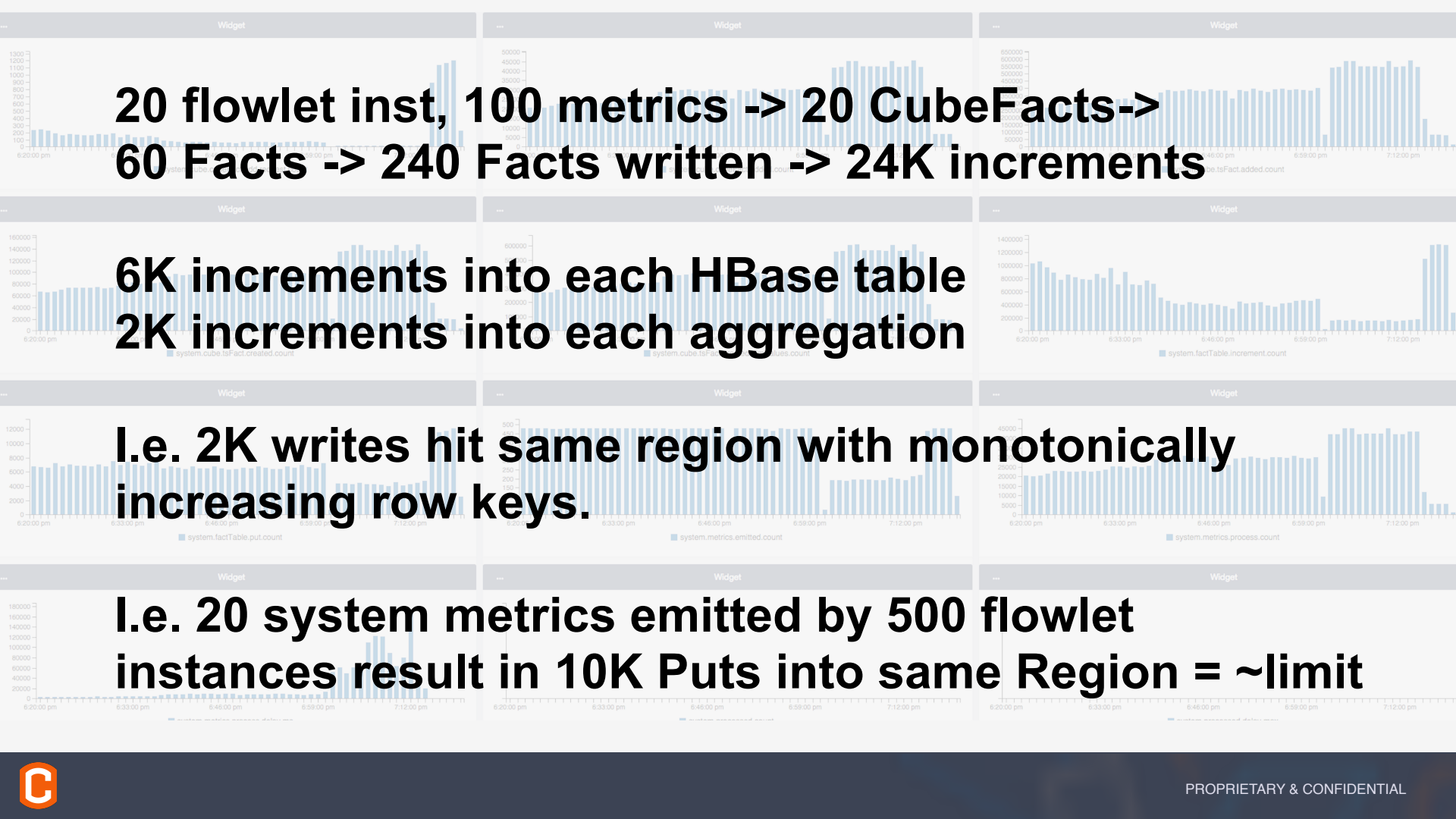
Each emitted metric may result into a LOT of HBase records written!

Note: within each table within each aggregation row keys are monotonically increasing, i.e. writes hit single region potentially creating a “hotspotting” issue.



Instrumentation!





**20 flowlet inst, 100 metrics -> 20 CubeFacts->
60 Facts -> 240 Facts written -> 24K increments**

**6K increments into each HBase table
2K increments into each aggregation**

**I.e. 2K writes hit same region with monotonically
increasing row keys.**

**I.e. 20 system metrics emitted by 500 flowlet
instances result in 10K Puts into same Region = ~limit**



Row key format:

```
<1-byte reserved for version>  
<encoded agg group><time base>  
<encoded dim1 value>...<encoded dimN value>  
<encoded measure name>
```

Note: no need to put dim name in row key, as agg group identifies dim list.

Number of HBase records written reduces when metrics processor grabs bigger batches of metrics data to process in single iteration

metrics contexts = # CubeFacts

Facts fanned out = # CubeFacts * # **aggregations**

Facts written = # Facts * # **resolutions**

HBase records written = # of measurements in Facts written



**50 flowlet inst, 100 metrics -> 5K* Puts into
same Region (60K* total) results in 2-5 sec
delay in processing**

**200 flowlet inst, 100 metrics -> 20K* Puts into
same region (240K* total) results in 120 sec
delay in processing**



50 flowlet inst, 100 metrics -> 5K* Puts into same Region (60K* total) results in 2-5 sec delay in processing

200 flowlet inst, 100 metrics -> 20K* Puts into same region (240K* total) results in 120 sec delay in processing

But it keeps up!



Verdict: lots of Opportunities!

- A single metrics processor may not be able to keep up (within reasonable processing delay) even with hundreds of program containers running on cluster.
- Cannot increase processing throughput by simply scaling out metrics processors.



Suggested Improvements

- No pre-split of Cube tables -> “slow start”
 - pre-split per aggregation?
- Hotspotting within aggregation -> hard limit writes, esp. in “bigger aggregations”, e.g. “across namespace”
 - salt rowkeys within aggregations?
 - + fix scalability issues?
- (?) Metric name in row key vs column key -> many “narrow Puts” vs few “wide Puts”
 - move measure name into column?
 - should only speedup queries
- Reduce number of aggregations, esp. bigger ones
 - may result in more complex querying: across aggregations

