CONTINUOUS AGGREGATIONS

06/12/2012 – VIRAL BAJARIA

BACKGROUND

- Realtime
  - Custom beacons
  - Batch Write to DB
  - 100s of millions of rows in each table
  - Custom reporting portal : ad hoc querying

- Batch Processing
  - 1 machine, multi threaded application
  - Processed logs received from third-party or internal beacons
  - Performed some aggregations (but still 10s of millions of rows)

BACKGROUND

- Problems
  - Ugly stored procedures to pull data
  - Runtime joins against metadata
  - Small resultset also needed to process millions of rows
  - Uniques was a nightmare!
    - 1-year into public launch took over 3 days of processing to do monthly
  - Reporting portals could never scale for data requests
  - Timezone shifting was a big problem (i.e. EST/PST)
    - Data was always in UTC
  - No single source of truth for data
  - Overall scale was a big problem

1ST SOLUTION

- Hadoop (What else??)
  - Will solve all problems ☺
  - First job was written in January 2009
  - Speed of processing was just mind-blowing
  - Write Map/Reduce jobs for each different cuts of data
  - In a year we had over 100s of jobs
    - Each job ran over the log data
- Problems
  - Raw logs had to be reprocessed
  - Outputs didn't match at different cuts
    - i.e. day/week/month
  - Timezone shifting was still an issue
  - Operational nightmare
    - Just 2 people on the team
  - Output was still written to DBs

2ND SOLUTION

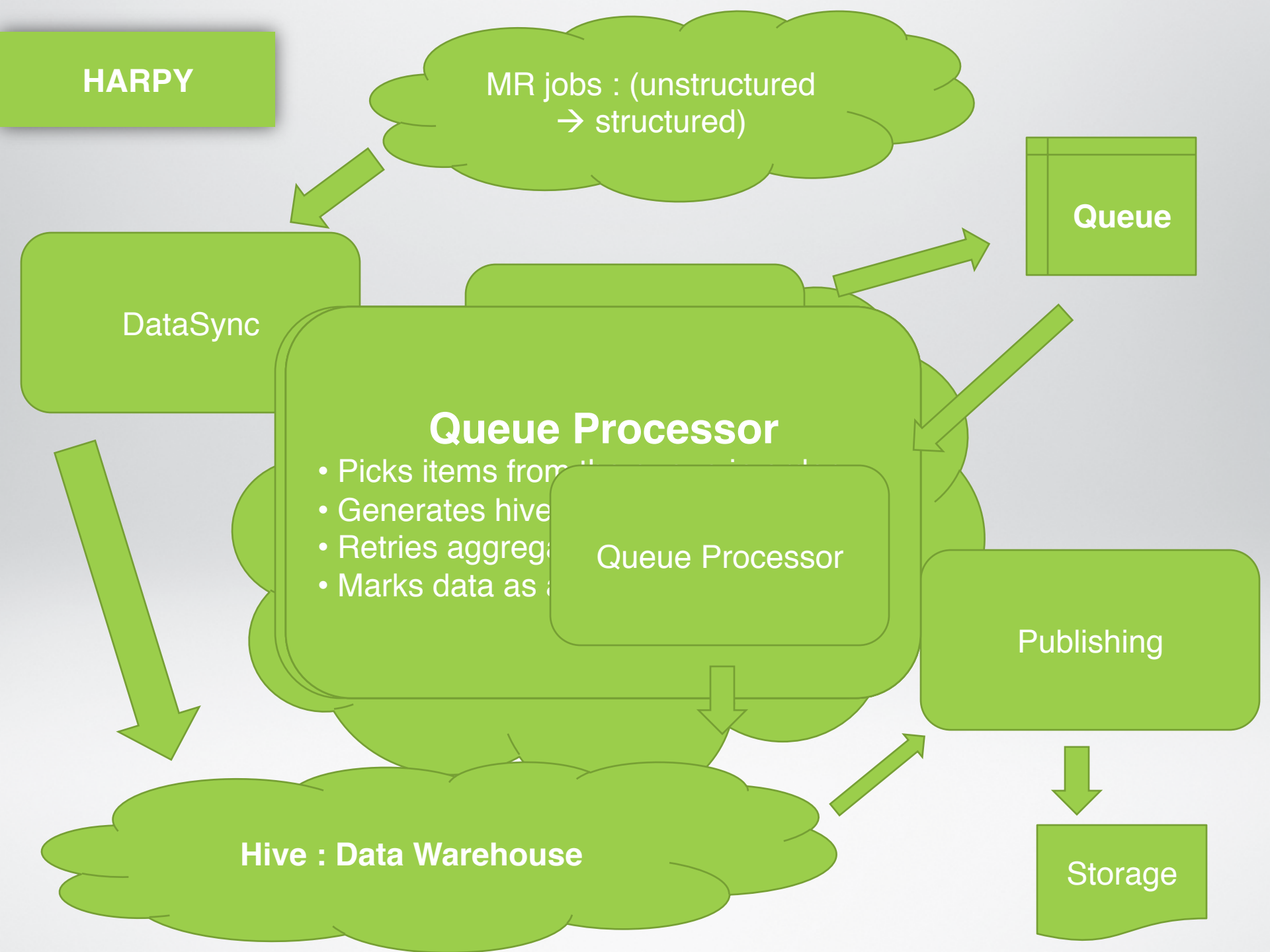
- Internal Name : Project Harpy
- Goals
  - Single source of truth (for a given fact)
  - Reduce operational overhead
  - Easy way to write map/reduce jobs
  - Support different aggregation functions
    - Sum
    - Count
    - Uniques
  - Support complex analysis
    - json data
    - Cohort analysis
  - Fast ad-hoc querying
- Solution
  - Hive + Harpy

HARPY : CORE CONCEPTS

- Metadata
  - Dimensions only
  - Tables with no facts
  - Examples
    - Video, Content Partners (Content)
    - Campaign , Flight, Creative (Advertising)
- Data
  - Tables that carry facts
    - Video_Starts_Hourly (UTC)
    - Advertising_Impressions_Hourly (UTC)
- Aggregations
  - Projections of fact tables
    - Video_Starts_Day_PST
    - Advertising_Impressions_Day_EST
- Publishing
  - Publishes Aggregations to target database tables
    - Currently supports MySQL + MS-SQL

HARPY : COMPONENTS

- Engine (API)
  - Data management
  - Process management
- DataSync
  - Controller : create/modify tables in hive
  - Sync : move data from DBs + Files into hive
- Aggregation
  - Controller : create/modify aggregated tables in hive
  - Scheduler
  - Query Generator
- Publishing
  - Files / Sql DBs / MySQL DBs
- Queue Processor
  - Serializes operation i.e. create table before running query
  - Maintain dependency chain
  - Guarantees atomic nature of data
    - Once an aggregation runs, it will never re-run


HARPY
MR jobs : (unstructured → structured)
Queue
DataSync
Queue Processor
• Picks items from
• Generates hive
• Retries aggrega
• Marks data as
Queue Processor
Publishing
Hive : Data Warehouse
Storage

HARPY : ENGINE

- define_data($definition)
- define_agg($definition)
- define_publishtask($definition)
- thread_action($name, $action)
- Request_aggregation($agg,$condition)
- submit_simple_query($query)

HARPY : DATA - METADATA

```
<data name="Video" type="db" source="Hulu.Video" sync="true"
    syncquery="SELECT Id, SeriesId, Title FROM Video">
    <dim name="VideoId" type="int32"/>
    <dim name="Series.SeriesId" type="int32"/>
    <dim name="Title" type="string" />
</data>
```

Sync Workflow

- Check for changes since last sync
- Validate references table, directly
- Store relevant information in the metadata table
- Queue table creation

Create Workflow

- Check for changes since last sync
- Validate references table, directly
- Store relevant information in the metadata to avoid duplicates (no updates in Hive)
- Queue
    - No runtime checking

```
CREATE TABLE IF NOT EXISTS Video
(
    VideoId int,
    SeriesId int,
    Title string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LINES
TERMINATED BY '\n' STORED AS TEXTFILE
```

HARPY : DATA – FACT

```
<data name="Video_Starts_Hourly" type="file" source="{directory-path}"
    sync="true" completeness="h">
    <dim name="hour.hourid" type="int64" partition="true" />
    <dim name="Video.VideoId" type="int32"/>
    <fact name="starts" type="int64" />
</data>
```

Create Workflow

- Check for name conflict
- Validate references to metadata tables are valid
- Store table configuration
- Execute CREATE TABLE command

Remove Workflow

- Check for table existence
- Check if referenced by other facts / aggregations (foreign key check)
- If all tests passed, then remove table info and drop table from Hive

HARPY : DATA – FACT

```
<data name="Video_Starts_Hourly" type="file" source="{directory-path}"
    sync="true" completeness="h">
    <dim name="hour.hourid" type="int64" partition="true" />
    <dim name="Video.VideoId" type="int32"/>
    <fact name="starts" type="int64" />
</data>
```

```
CREATE TABLE IF NOT EXISTS Video_Starts_Hourly
(
        VideoId int,
        Starts bigint
)
PARTITIONED BY ( hourid bigint )
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LINES
TERMINATED BY '\n' STORED AS TEXTFILE
```

HARPY : QUERY GENERATION

- Retrieve item from queue
  - Eg.: video_starts_day_estl(hourid>=372077 and hourid <= 372100)
  - Aggname : video_starts_day_est
  - Condition : 24 hours of data
  - Source table : retrieved from agg definition
- Build list of columns in SELECT and GROUP BY
- Build list of tables
  - Fact tables
  - JOIN tables
  - Global map which maintains table-name → alias map
- If partitioned,
  - Remove from select and group by
  - Insert the partition information
- Build WHERE condition
  - state machine to validate the where condition
  - sanitizer to perform table alias
- Generate hive-compatible QUERY

HARPY : AGGREGATION (SIMPLE)

```
<agg name="Video_Starts_Day_EST" completeness="d" timezone="est">
    <dim name="hour.est.real_date" type="string" partition="true"/>
    <dim name="VideoId" type="int32"/>
    <sum name="starts" type="int64" data="video_starts.starts" />
</agg>
```

```
INSERT OVERWRITE TABLE Video_Starts_Day_EST ( real_date =
“2012-06-12” )
SELECT /*+ STREAMTABLE(t1) */
        t1.VideoId, SUM(t1.Starts)
FROM
        Video_Starts t1
WHERE
        HourId >= 372077 AND HourId <= 372100
GROUP BY
        t1.VideoId
```

HARPY : AGGREGATION (JOINS)

```
<agg name="Series_Starts_Day_EST" completeness="d" timezone="est">
    <dim name="hour.est.real_date" type="string" partition="true"/>
    <dim name="Series.Title" type="string"/>
    <sum name="starts" type="int64" data="video_starts.starts" />
</agg>
```

```
INSERT OVERWRITE TABLE Series_Starts_Day_EST ( real_date =
“2012-06-12” )
SELECT /*+ STREAMTABLE(t1) */
        t2.Title, SUM(t1.Starts)
FROM
        Video_Starts t1
        JOIN Series t2 ON t1.SeriesId = t2.SeriesId
WHERE
        HourId >= 372077 AND HourId <= 372100
GROUP BY
        t1.VideoId
```

HARPY : PUBLISHINGS

```
<publish name="Series_Starts_Day_EST" desttype="sql" destdb="{db-
    name}" desttable="{table-name}" aggname="Series_Starts_Day_EST">
    <map destColName="real_date" destColType="datetime"  aggColName="real_date"/>
    <map destColName="SeriesTitle" destColType="string"  aggColName="Title"/>
    <map destColName="total_count" destColType="bigint" aggColName="starts" />
</publish>
```

HARPY : UNIQUES

```
<data name="Video_Starts_Hourly" type="file" source="{directory-path}"
    sync="true" completeness="h">
    <dim name="hour.hourid" type="int64" partition="true" />
    <dim name="Video.VideoId" type="int32"/>
    <fact name="{user-identifier}" type="string" />
    <fact name="starts" type="int64" />
</data>
```

```
<agg name="Video_Uniques_Week_PST" completeness="w">
    <dim name="hour.est.week_end_date" type="string" partition="true" />
    <dim name="Video.VideoId" type="int32"/>
    <unique name="uniques" type="int64" data="Video_Starts_Hourly.{user-
    identifier}" />
</agg>
```

HARPY : UNIQUES

```
<data name="Video_Starts_Hourly" type="file" source="{directory-path}"
    sync="true" completeness="h">
    <dim name="hour.hourid" type="int64" partition="true" />
    <dim name="Video.VideoId" type="int32"/>
    <fact name="{user-identifier}" type="string" />
    <fact name="starts" type="int64" />
</data>
```

```
<agg name="Total_Uniques_Month_PST" completeness="m">
    <dim name="hour.pst.month_end_date" type="string" partition="true" />
    <unique name="uniques" type="int64" data="Video_Starts_Hourly.{user-
    identifier}" />
</agg>
```

HIVE : AD-HOC QUERIES

- Harpy (submit-simple-query)
  - submit-simple-query api
  - Takes a pre-built hive query and runs by connecting to an existing hive thrift server port
  - Blocking call, can't execute multiple queries
- web service
  - Written in python, uses tornado
  - Takes a pre-built hive query
  - Returns results in json or xml format
    - Easily integrates with reporting and analytical services
  - Dynamically opens hive ports for query execution
    - Ports are re-used across a thread pool
    - Helps achieve parallelization

HIVE : COHORT ANALYSIS

- Cohort Generation
  - Custom or templated hive queries
  - Generates a list of userids
  - Based on demographic, subscription and usage behavior
  - Output stored into hive table
    - Partitioned on cohort name
    - Userid and Timeperiod
  - Users can be "tagged" with cohort for online usage in HBase

- Cohort Usage
  - Cohort table joined against usage information for historical analysis
  - Online usage
    - Ad targeting
    - Marketing campaigns
    - Recommendations
- Queries submitted via the submit-simple-query framework of Harpy

HARPY : FUTURE WORK

- Filtering
  - Allow <filter> tags during table definition
- Custom JOIN conditions
- Progressive aggregations
  - week-to-date
  - month-to-date
  - trailing-30-days
- Aggregations over JSON column

HARPY : REASONS

- Integrated nicely with our DB based approach
  - Entire reporting stack ran off SQL tables
- Started off with 10s of tables
  - Currently we run 1000s of aggregations
  - We don't have to manage all that data
- Humans don't scale
  - Query generator helps write map/reduce jobs
- Timezone shifting without any overheads
  - We can do UTC/EST/PST/JST
- None of the existing tools were mature enough at that time
  - Started in 2009, Sqoop was not even around
  - Hive was in version 0.3 (we went into production with 0.5)
- Helped scale out our metrics platform with just 2 people on the team
  - We are at 5 now ☺

THANK YOU

QUESTIONS ?
WE ARE HIRING IN SILICON BEACH