"If everything seems under control, you're just not going fast enough …"

-Mario Andretti

Designing a High Speed GDS Telemetry Repository using Open Source Technologies

Lloyd DeForrest, *Jet Propulsion Laboratory, California Institute of Technology*
Farzad Saadat, *Jet Propulsion Laboratory, California Institute of Technology*
Philip Southam, *Jet Propulsion Laboratory, California Institute of Technology*

Approved for US and foreign release: CL# 16-0585

Briefing Overview

- Task Overview
- JPL's Telemetry and Command System Overview
- Task Concept Diagram
- Front Line Data Pipeline
- Stream Processing
- Persistent Storage
- Putting Them All together
- Future Work
- References/Contacts

Task Overview

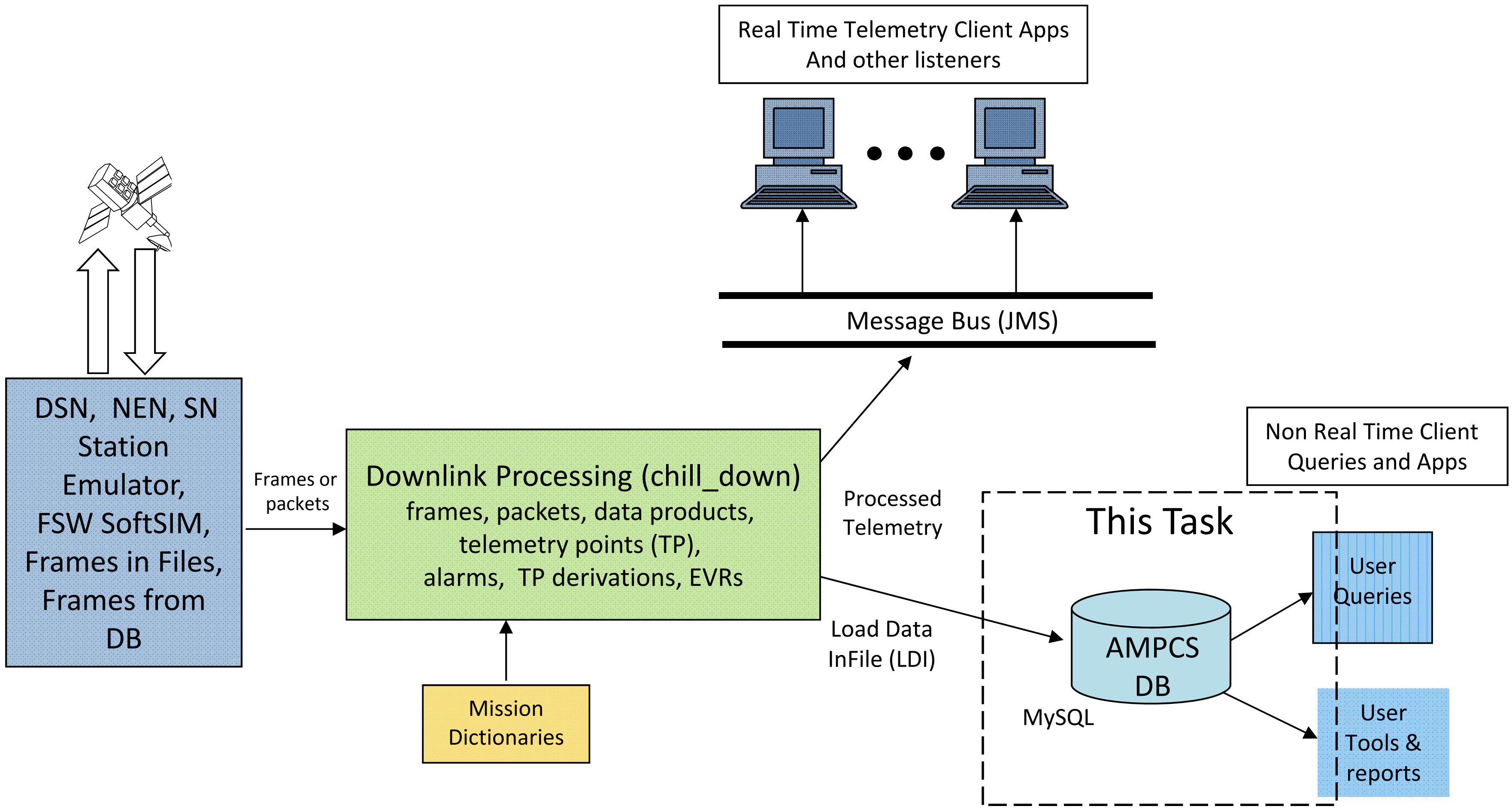
- JPL's current Telemetry and Command System was developed for the Mars Science Laboratory mission beginning in 2006-2009
- Supports 5 missions with many more coming
- It uses MySQL has its persistent data store
    - A highly indexed "query" database for query fast response times
    - A lightly indexed "load" database for low latency telemetry loading (for large missions)
    - The load database prevents backing up the upstream processing
  - Once established in a mission, MySQL does not scale
    - Query times get slower as the datasets get larger
    - Hard to modify database schemas
  - The necessary use of "joins" further increases query response times

- Our task is to completely replace the current MySQL based implementation with:
  - A modern equivalent that takes advantage of distributed, parallel processing to achieve orders of magnitude improvements in query responsiveness
  - A modern equivalent that is easier to extend and scale up or down as the need arises
  - A modern equivalent that can satisfy the needs of a small missions, such as CubeSats and large-scale flagship missions
  - Instances that are free of license fees

JPL's Telemetry and Command System Overview

- Advanced Multimission Operations System data Processing and Control System (AMPCS)
  - Telemetry Input as CCSDS Frames and/or Packets
  - Processing to create Event Records (EVRs), Telemetry Points (TP) , TP Alarm Notification, TP DN to EU enumeration, Derived TPs and Product Building
  - Command Handling and various levels of automation and reporting
  - All of the above products (including frames and packets) are stored in a MySQL database for the Life of Mission

AMPCS Overview (Current State)



Note: Uplink components not shown

Task Concept Diagram

User Applications (Web and CMD line I/Fs)

Persistent Data Storage
- Long Term Storage
- Cloud or VM based
- High Performance, Low Latency
- Easy to deploy to multiple sites
- No License fees

Area of study

Real-Time Stream Processor
- Orchestrates multiple stream processing
- Prepares for loading to the Persistent Store

Area of study

Front-line Data Pipeline
- Ingests/buffers incoming streams
- Prepares data for stream processor

Area of study

chill_down → EVRS, Alarms → Front-line Data Pipeline

chill_down → Frames, packets → Front-line Data Pipeline

chill_down → Telemetry points → Front-line Data Pipeline

Front-Line Data Pipeline Overview

- Accepts incoming telemetry streams from chill_down
  - High rate telemetry point data, EVRs, Alarms, DN-EU data, frames, packets, etc.
- Buffers the incoming streams data, allowing the chill_down process to run at full rates.
- We choose Apache Kafka to satisfy the needs for the Front-Line Data Pipeline.

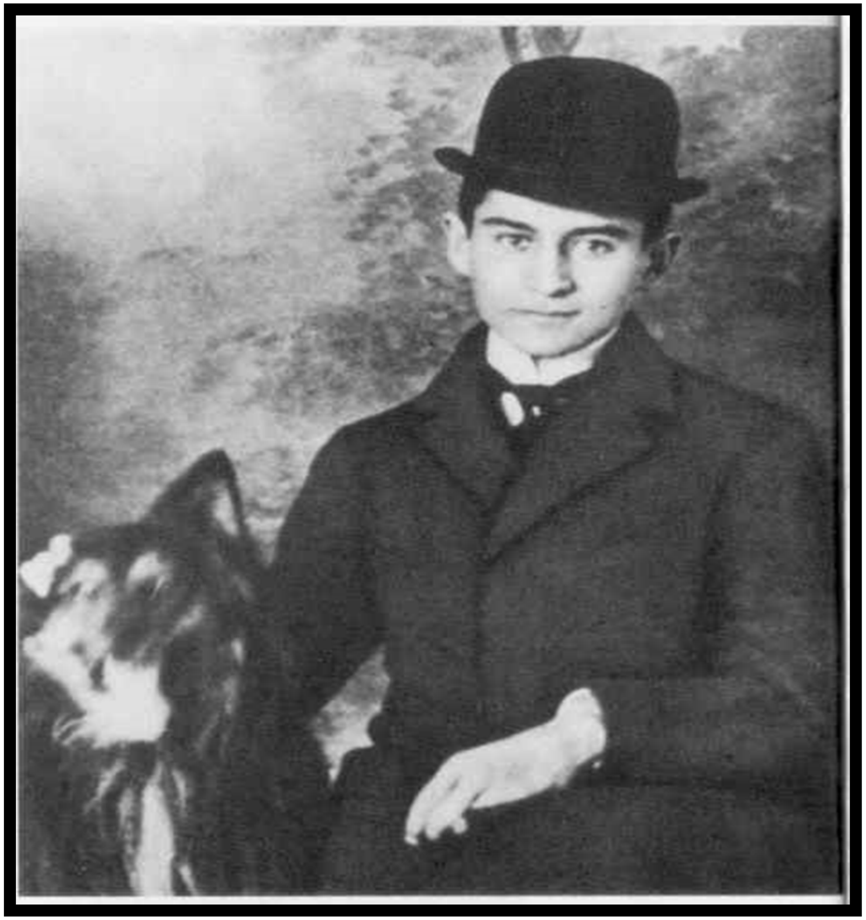
- Why did we choose Kafka?
  - Kafka is an Apache project.
  - Kafka is a distributed, high throughput and scalable **publish-subscribe based messaging system**.
  - Kafka separates the data source (chill_down) from the stream processing layer. (**separation of concerns**)
  - Kafka separates raw data and writes them into topics, then aggregates them while maintaining the sequences. For each topic, Kafka allows partitioning to fully utilize **parallelism in the cloud**.
  - Compared with traditional messaging systems, such as ActiveMQ or RabbitMQ, Kafka has much better throughput and built-in partitioning,
  - **Replication and fault-tolerance** make it a good solution for large scale message processing applications.

Kafka

Franz Kafka:
1883-1924

But we are
talking about:

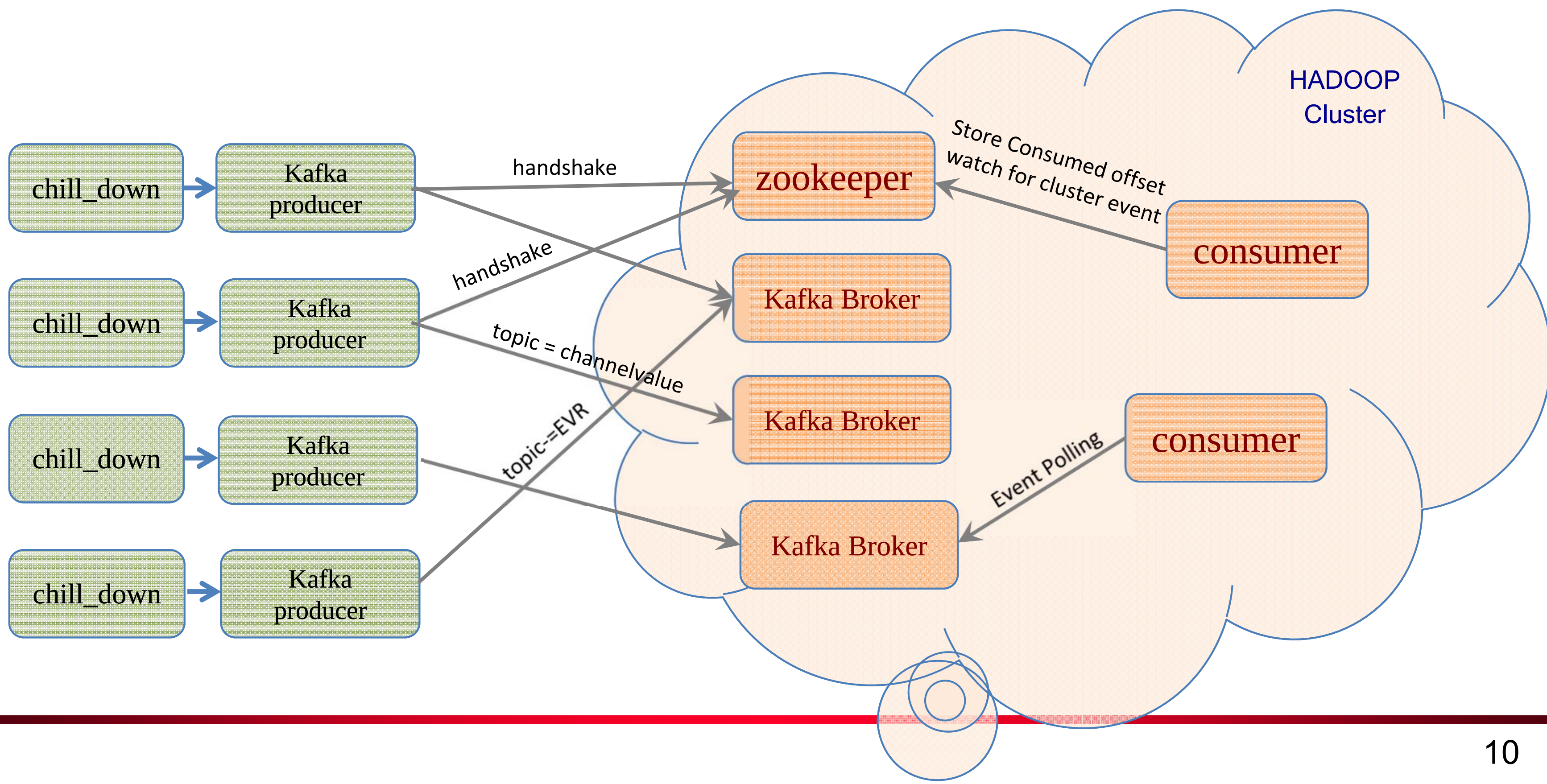
Apache Kafka
2011-2016



Front-Line Data Pipeline Design

Childown - Kafka Integration

We have modified chill_down to create streams of telemetry types



Stream Processing Overview

- Next is the real-time stream processing framework that serves as the consumer for Kafka.
  - Prepares and orchestrates high speed loading of telemetry products to the persistent data store
  - Must integrate well with Kafka and the persistent data store
- Apache offers a handful of frameworks for this purpose
  - Our challenge is to pick the one that fits our system specifications and demands.
- We prototyped the following frameworks for our system:
  - Spark
  - Flume
  - Storm
  - Samza

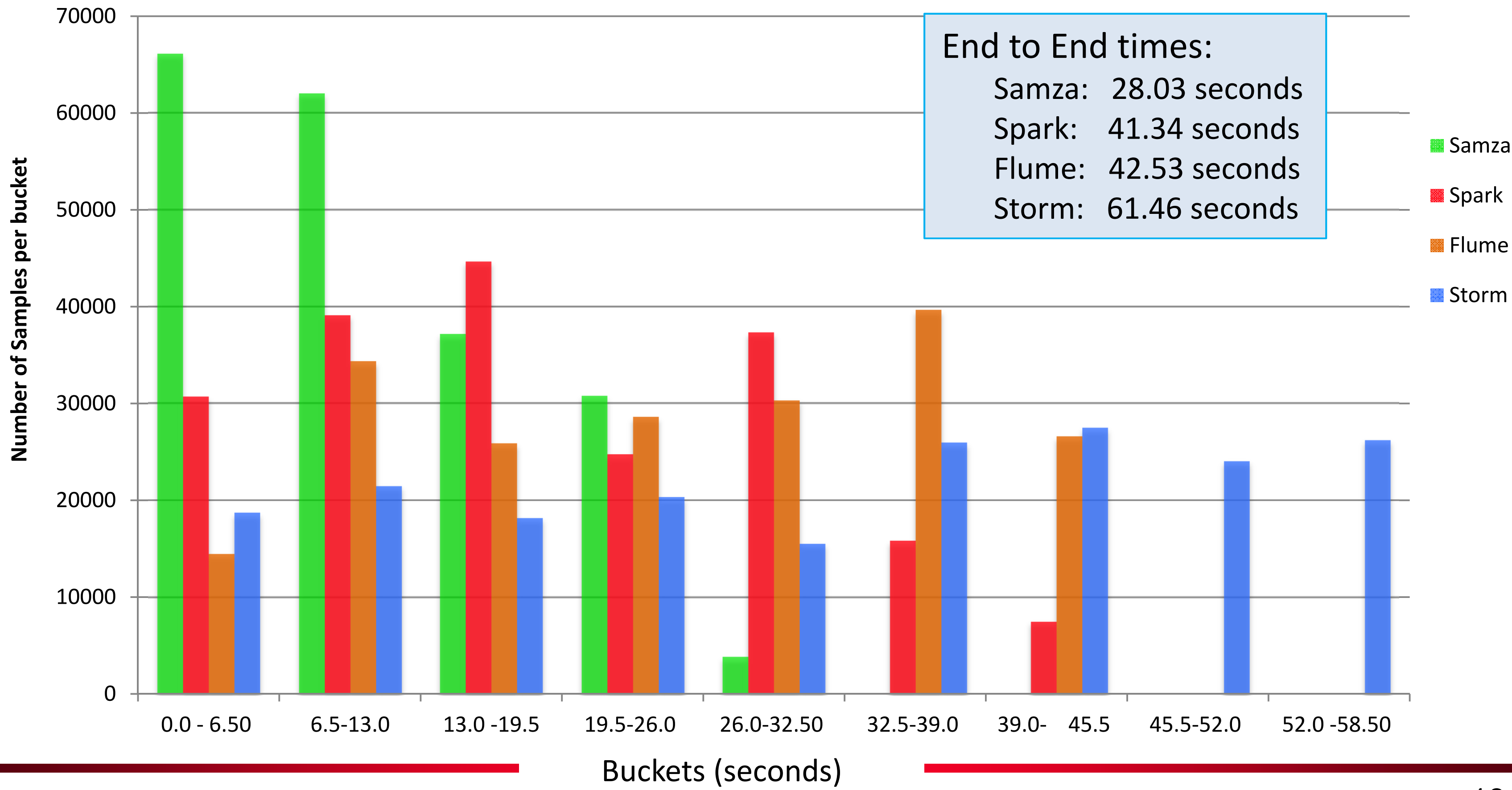
Stream Processing Component Selection

- All candidates are fast, horizontally scalable, fault tolerant, easy to operate and programming language agnostic and have guaranteed data processing and delivery
- Subtle differences makes one preferred over the others based on the application's design specifications.
- Comparison Matrix :

Apache	Spark	Storm	Samza	Flume
Stream source	Receivers	Spout	Consumers	Source
Data unit	D Stream	Tuple	Message	Event
Processing unit	Transformations Window Ops	Bolts	Tasks	Agent
Latency	Seconds (depends on batch size)	Sub-Second	Sub-Second	Sub-Second
State management	Stateful - Write state to Storage	Stateful with Trident	Stateful - Embedded key-value store	None
Language support	Java, Scala, Python	Any	Java, Scala	Java

Stream Processing Benchmark Results

- ✓ Benchmarking: We took 200,000 samples of Telemetry Point values and fed them to a 6-node Kafka cluster. As a Kafka consumer, we benchmarked each framework, for end to end processing time (output of chill_down to a test instance of Hbase).



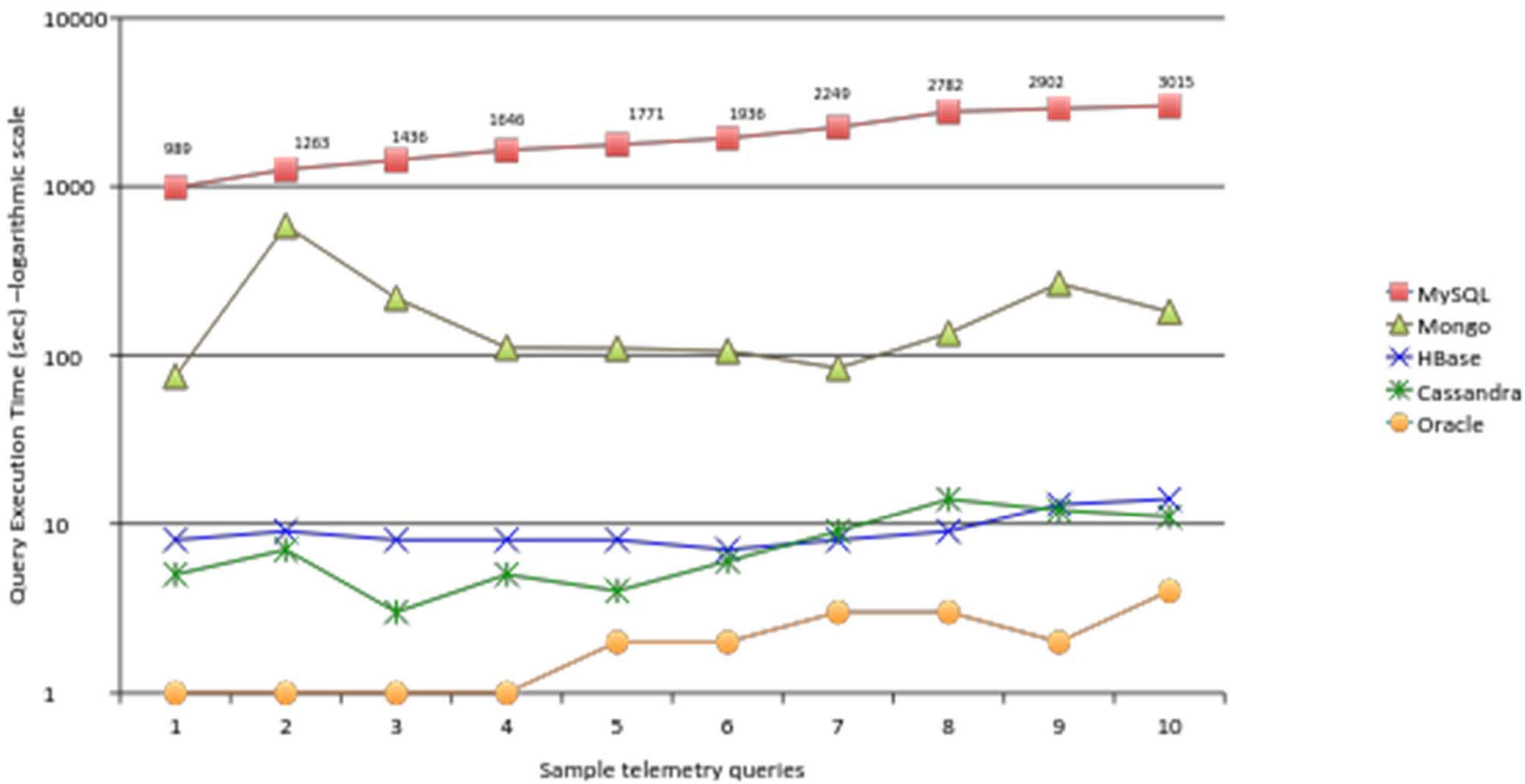
Apache Samza… for Real-Time Steam Processing

- Samza was selected because:
  - Highest performance overall
  - Full compatibility with Kafka (both Kafka and Samza were created by LinkedIn)
  - A rich set of APIs for Kafka and other web apps (e.g. websockets, etc)
  - Strong community support and an agile development team.
- Unlike Storm that uses a topology for its real time computations and master/worker node relations between nodes, Samza streams by processing messages as they come in, one at a time. The streams get divided into partitions that are an ordered sequence where each has a unique ID.
- Persistent state for durability and real time aggregations
- Uses Kafka as a message bus, making it easier for other applications to consume intermediary data products.

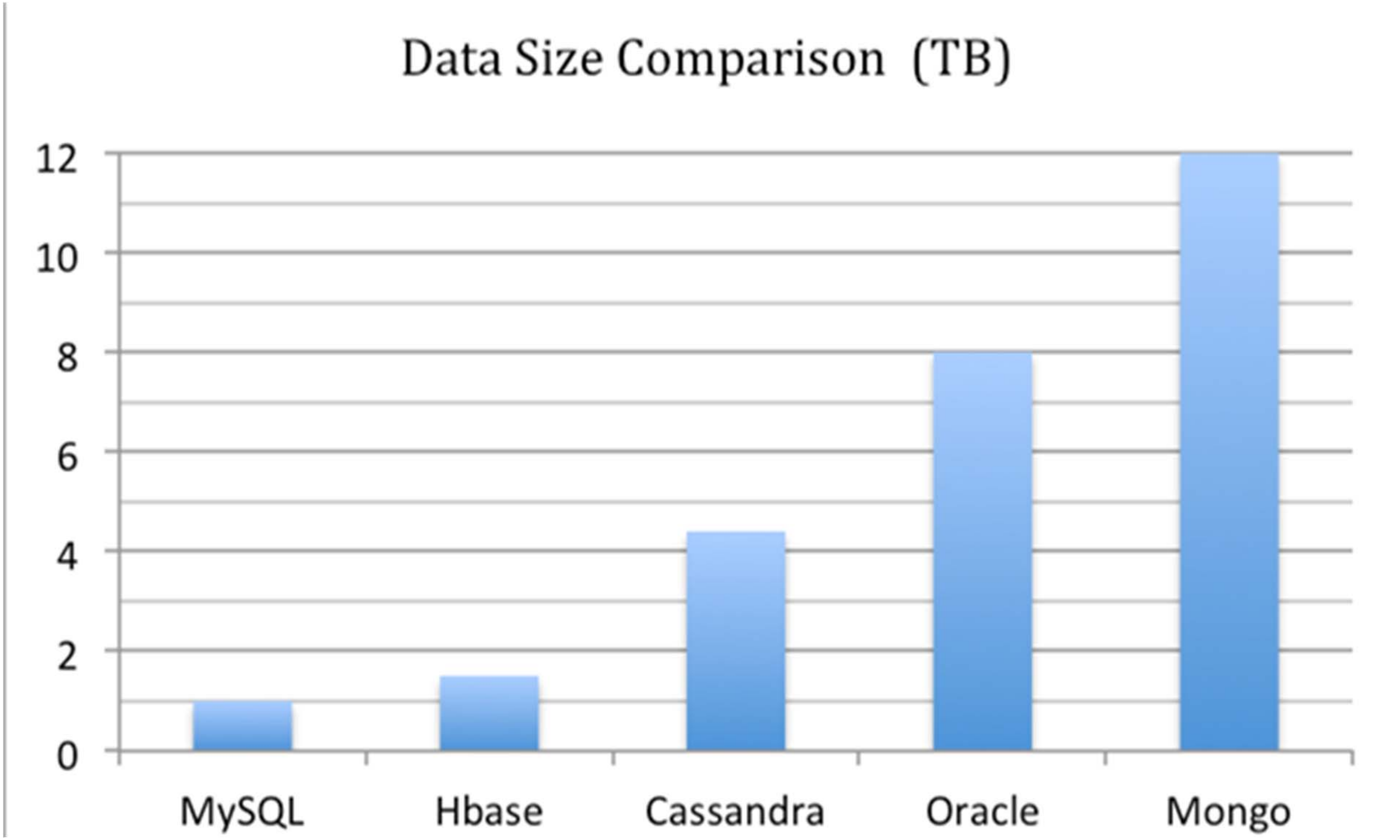
Persistent Storage Overview

- Next we benchmarked different technologies to persist our telemetry
- We investigated NoSQL databases for their high performance and extensibility
- We considered and prototyped the following technologies:
  - MongoDB
  - HBase
  - Cassandra
  - Oracle NoSQL
- Benchmark Procedure: We fed telemetry from a recent JPL mission (SMAP) into a prototype instance of each technology
  - Benchmarked 10 different types of queries
  - ~3B telemetry points stored on the reference MySQL instance, with a table size of approximately 1 TB

Persistent Storage Benchmark Query Results



Persistent Storage Benchmark Data Size Comparison

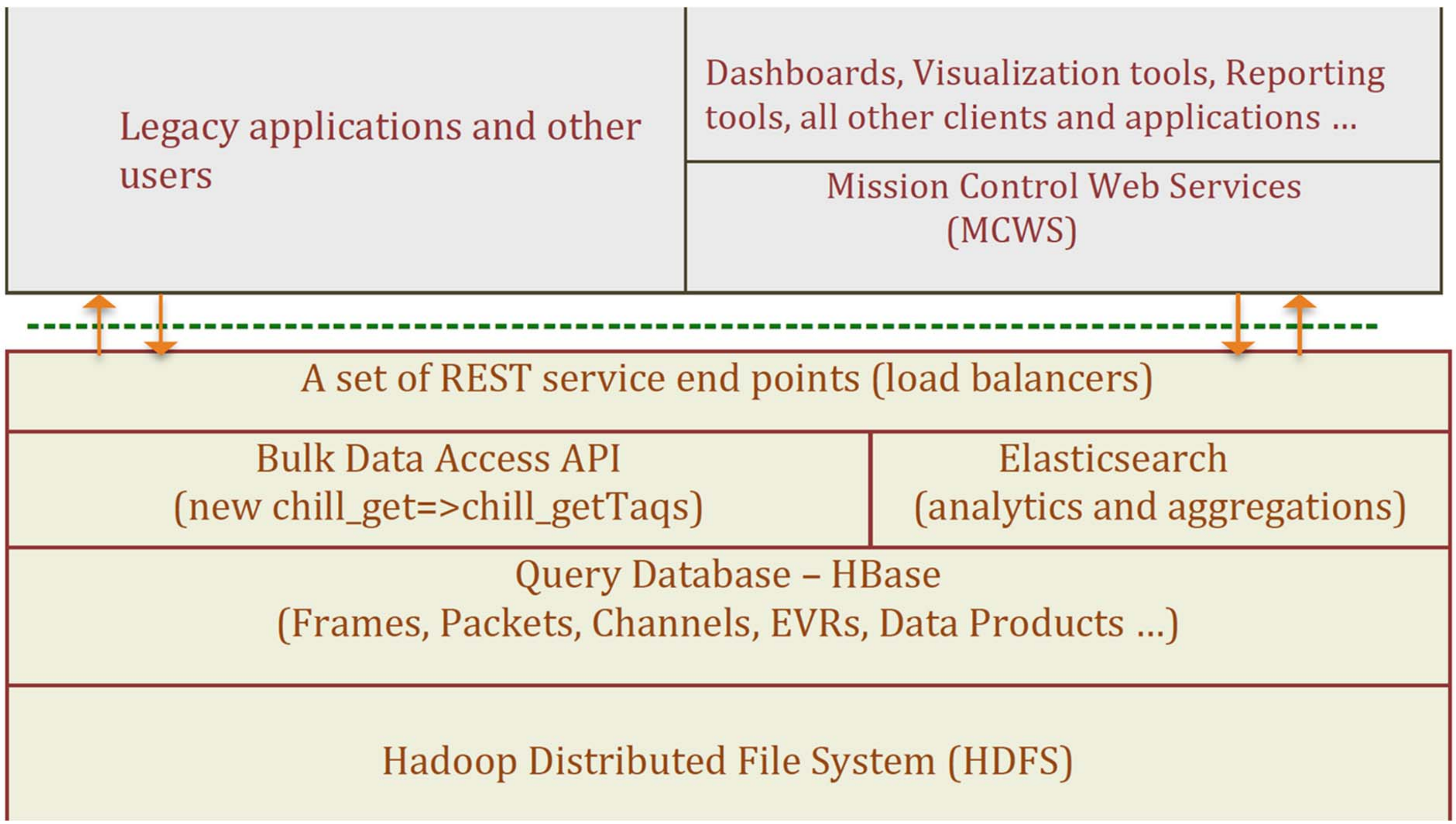


HBase and Elasticsearch

- We chose HBase as our database due to:
  - Relatively small data storage requirement
    - The entire table containing the dataset was about 500GB (total of 1.5TB for the original data plus 2 replicas).
  - Queries finished in seconds (orders of magnitude faster than MySQL)
  - Sharding in HBase is automatic (auto-sharding and auto-failover).
  - HBase compacts the data
  - HBase is the Hadoop database for real-time and random read and write access
  - It is massively scalable and can handle billions of records

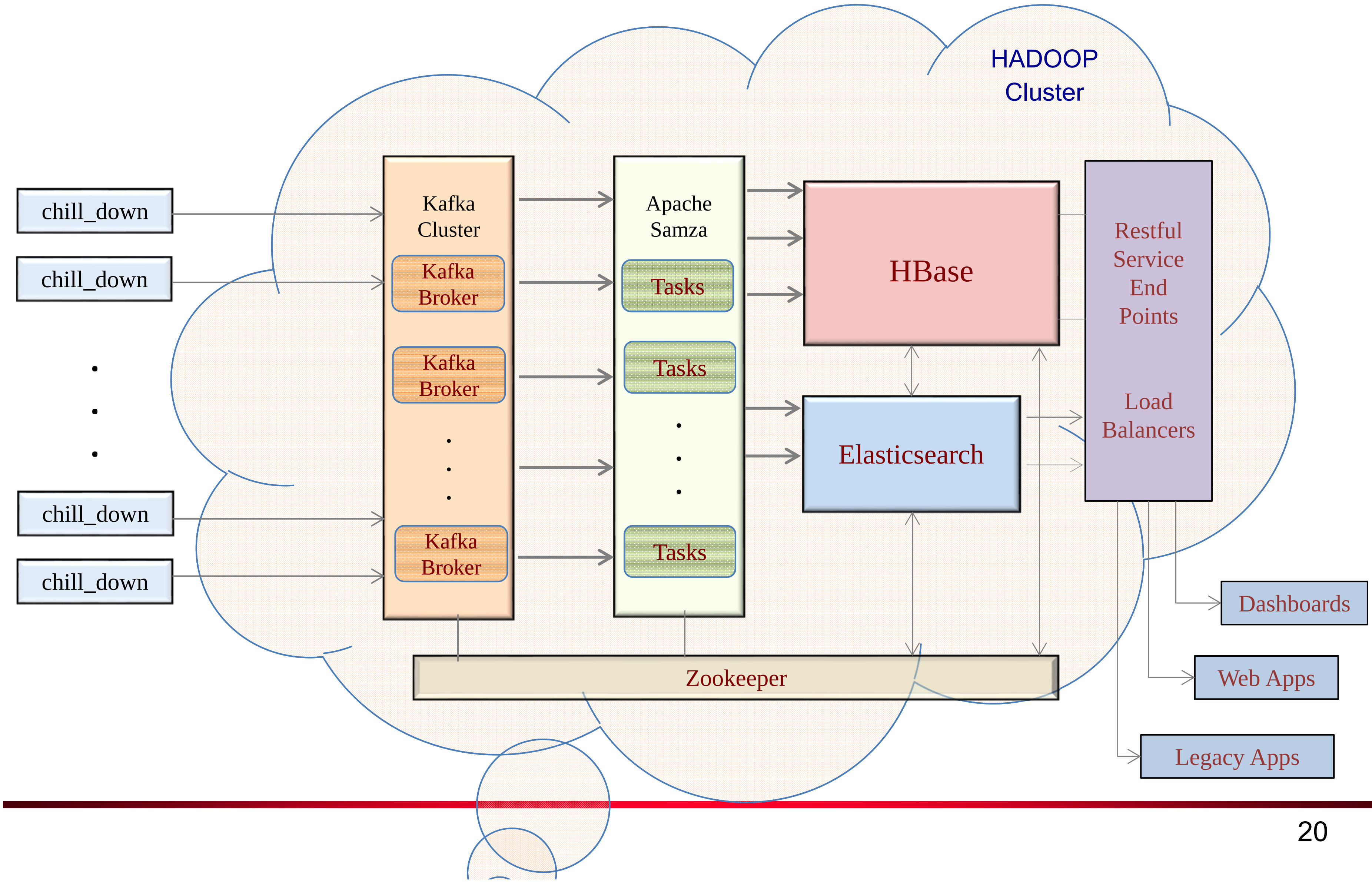
- While performing this study, we also chose to include Elasticsearch for data analytics and aggregations
- Elasticsearch features:
  - Flexible and powerful open source, distributed real-time search and analytics engine for the cloud
  - High availability, multi-tenancy, full text search, document oriented, conflict management, schema free
  - Offers two types of aggregations : Bucket and Metrics on our telemetry products

Persistent Storage Design



Telemetry Analytics and Query Stack (TAQS)

Putting Them All together



Future Work

- Finish End to End Prototype and Design
- Engineer for all supported venue profiles
  - Small Mission (single processor)
  - Medium scale (multiple VM)
  - Large Scale (multi-node cloud instances)
  - External Mission (e.g. University Cubesat)
- Replication (multi-node processing and storage)
  - Same data at the Launch facility, JPL Mission Support Area and remote sites
- Utilities
  - Installation and Upgrade tools
- Web-Interface layer interfaces
- Incorporate into the AMPCS baseline
  - Execute Formal Test Program

References/Contacts

References

- Apache Kafka: http://kafka.apache.org
- Apache HBase: https://hbase.apache.org
- Apache Storm: http://storm.apache.org
- Apache Samza: http://samza.apache.org
- Elasticsearch: http://www.elasticsearch.org

Contacts:

Lloyd DeForrest: lloyd.deforrest@jpl.nasa.gov

Farzad Saadat: farzad.saadat@jpl.nasa.gov

Philip Southam: philip.southam@jpl.nasa.gov

The work described in the briefing was carried out at the Jet Propulsion Laboratory (JPL), California Institute of Technology (Caltech), under a contract with the National Aeronautics and Space Administration (NASA). The work was funded through the Multimission Ground System and Services Office (MGSS) at JPL.

Questions?