TU Delft

Delft University of Technology

# Scalable Data Processing System for Satellite Data Mining

Speretta, Stefano; Ilin, Anatoly

**Publication date**
2017

**Citation (APA)**
Speretta, S., & Ilin, A. (2017). *Scalable Data Processing System for Satellite Data Mining*. Abstract from 68th International Astronautical Congress: Unlocking Imagination, Fostering Innovation and Strengthening Security, IAC 2017, Adelaide, Australia.

**Important note**
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

**Copyright**
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

**Takedown policy**
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

*This work is downloaded from Delft University of Technology.*
*For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.*

68th International Astronautical Congress (IAC), Adelaide, Australia, 25-29 September 2017.
Copyright © 2017 by the International Astronautical Federation (IAF). All rights reserved.

IAC-17-B1.6.6

# Scalable Data Processing System for Satellite Data Mining

**Stefano Speretta[a*], Anatoly Ilin[b]**

[a] *Delft University of Technology (TU Delft), Kluyverweg 1, 2629 HS Delft, The Netherlands,* s.speretta@tudelft.nl
[b] *Delft University of Technology (TU Delft), Kluyverweg 1, 2629 HS Delft, The Netherlands,* a.ilin@student.tudelft.nl
[*] Corresponding Author

**Abstract**

This paper describes the development of a new ground station infrastructure targeted towards massive swarms and constellations. Mission trends are first analyzed to derive the design drivers for such a system and then the general architecture is analyzed. The target architecture, based on "Big Data" processing architectures is presented, clearly showing how to re-use current data processing state-of-the-art systems for satellite operations. This paper also describes the ongoing developments to integrate standard data mining and artificial intelligence software frameworks in the data acquisition system to develop a complete system capable of acquiring data from multiple sources, autonomously process them and deliver them to users.

**Keywords:** CubeSat, swarm, constellation, ground station, data mining

## 1. Introduction

Satellite swarms and constellations are becoming more and more widespread thanks to hardware and launch cost reduction. Nano-satellites are proving extremely suited for such big constellations where the single satellite has very limited capabilities but, when combined, very powerful systems could be created (capable for example of observing the whole Earth once a day). One of the problems arising from this trend is the constant increase in data to be transmitted to ground and the increased complexity in running a constellation with more than 100 satellites.

Several institutions have invested consistent effort in the development of more capable ground systems to acquire and process all the data. The geographical distribution of such infrastructure is becoming critical and so does the capability of aggregating data from multiple sources (the spacecrafts): this infrastructure is getting more and more similar to the one used by most web companies (like Facebook, Google, etc...) to process analytics coming from web pages. Both infrastructures need to swallow big amounts of data (or "Big Data") in quasi-real-time: in both cases further data analysis (or mining) could help identifying hidden trends (such as, for example, possible failures). All these points lead to the design of this system, trying to re-use part of the existing data processing infrastructure and applying it to satellite data analysis. By coupling together a data-gathering section (acquiring data from the different ground stations) and a data analysis section, we aim at developing a full ground system capable of supporting massive constellation, up to the point where human operators would have problems running it.

In the following sections we will analyze the current trends in satellite systems (see Section 2 to justify our attention towards massive constellations). We will then analyze the architecture of such system by starting from the experience gathered with nano-satellites and looking at extending it to big constellations (see Section 3). Proposed Lambda architecure will be presented (see Section 4), together with some future work (see Section 5) and conclusions (see Section 6).

## 2. Space Mission trends

The steep decrease in launch cost in the past decade lead to an increase in satellite launches per year, as also shown in Figure 1. In particular, nano-satellites saw a tremendous increase in launches per year (264 CubeSats were launched in the first 3 quarters of 2017, as compared to the total number satellites launched between 2015 and 2016)[2]. But unfortunately the performances of a single satellite are still limited when compared to big-

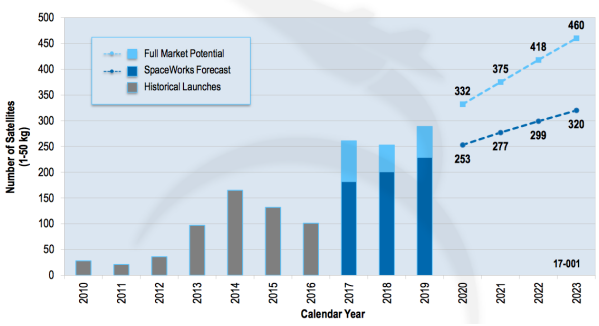


Fig. 1: Nano/Microsatellites launch trends[1].

Copyright © 2017 by the International Astronautical Federation (IAF). All rights reserved.

ger missions and this lead, together with the dramatic launch cost reduction, to the diffusion of multi-satellite missions[3][4], that are also being proposed for several purposes, like climate science [5][6], atmospheric observations [7][8][9] or disaster monitoring[10]. At the same time, constellations and swarms are becoming as big as 197 satellites for Earth observation, such as the Flock constellation from Planet, whose goal is imaging the whole Earth at coarse resolution once a day[11].

The ground infrastructure for a massive swarm needs to handle a high number of satellite passes per day (approximately 350 with a fleet of 50 satellites[12], expected to grow to 1400 with the full constellation of approximately 200 satellites) from different locations worldwide. Multiple locations would be fundamental to achieve such goal (see Figure 2, for example) and a strong and fast data processing network will be fundamental too.

A lot of research going into massively distributed ground systems, as can be seen in [14][15][16][17]. But beside the pure data collection and archiving, handling massive amounts of data poses challenges in itself (system scalability, flexibility and fault tolerance) that are currently being addressed. Operations of such big swarms and constellations proves also critical, having the operators a huge number of satellites to monitor and control. All these reasons lead us to study the architecture of a massively distributed ground infrastructure, which will be presented in the next sections.

## 3. System architecture evolution

### *3.1 The legacy, no scalability*

As a starting point of the discussion and to a familiarize the reader with the core framework functionality, consider the legacy implementation outlined in Figure 3. Telemetry processing system has been developed for an educational nano-satellite mission (Delfi-n3Xt): relying on the Delft ground station, as well a set of third party radio-amateurs submitting data through an ad-hoc client

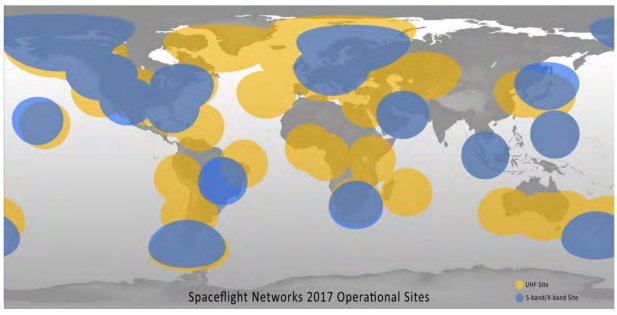


Fig. 2: Spaceflight distributed ground station network[13].

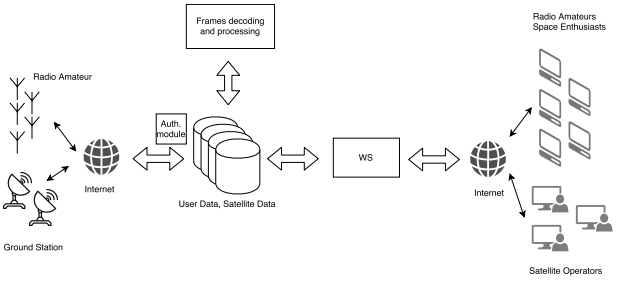


Fig. 3: Delfi Legacy architecture[21].

application [18]. The client application performed demodulation, decoding and limited data visualization [19]. Upon successful client authentication, received satellite data (data frames[20]) is injected directly to the SQL database on the Delfispace server. A set of processing scripts, controlled by a scheduler, decoded binary data frames to set of satellite metrics and store the latter in the database. Finally, the satellite functional parameters, filtered by client permissions, are made accessible to the stakeholders via a simple web server[21]. The legacy system in Figure 3 is a classic example of a "monolithic architecture" [22]. The limited scalability expresses itself in two ways: data processing and server scalability. Processing is limited to a single data frame definition and cannot be extended. Server scalability can achieved vertically, by allocating more resources, or horizontally, by running additional servers in parallel. In practice, horizontal scalability is preferred due to redundancy concerns. Arguably, aforementioned can be accomplished by deploying the legacy database in load-balanced configuration [23], with additional monitoring of processing scripts and load balanced web servers. This effort will grossly under perform compared to purpose built scalable systems [24][25].

### *3.2 Scale by leveraging clients*

It should be noted, that in the case of Delfi-n3Xt mission, clients submitting data frames, simultaneously acts as clients retrieving processed data. A possible evolution of the previous system is shown by Figure 4, where the system relies entirely on a distributed database system for data transport and it is based on the unique server-side Couch DB and client-side Pouch DB ecosystem. PouchDB is a javascript implementation of CouchDB, a no-SQL, document database with out-of-the-box enabled sharing and data replication capability[26][27]. Being written in javascript, PouchDB runs in the web browser, serving database-stored web pages and performing data visualization, even while offline. By deploying this ecosystem, satellite data and web pages can be replicated to the clients, reducing the load on the central server. Any newly received satellite data by any ground station will

Copyright © 2017 by the International Astronautical Federation (IAF). All rights reserved.

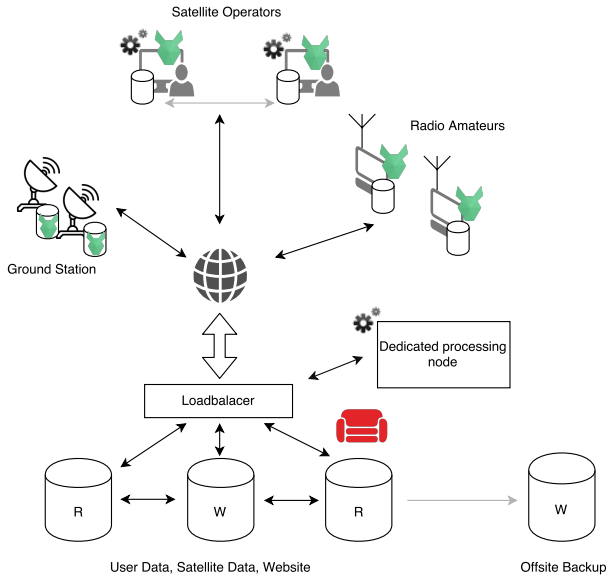


Fig. 4: Proposed client cluster architecture.

be replicated to the central server and replicated to all the clients.

To increase system stability, a load balancer is introduced and configured to perform write operations only to one node, and "read" on two dedicated nodes.

This correlates to the CAP theorem[28]: any data storage system can only ensure two out of three characteristics: Consistency, Availability or Partition Tolerance. Originally proposed by Marz[29], as a solution to reduce the system complexity by "the use of mutable state in databases and the use of incremental algorithms to update that state", provides a solution to the problem. This facilitates the partition tolerance (P) and availability (A). CouchDB[30], like many other no-SQL databases, prefers A and P over the Consistency (C), meaning that at a particular time after new data ingestion or data change, there will be nodes serving different versions of the data [28]. Hence the terminology "eventual consistency".

The choice of a single "write" and double "read" node is not arbitrary. Architecture is by design "read" centered, demanding a low latency for data replication to the clients. As a bonus, this contributes to the higher availability. The single "write" node eliminates issues with document duplication on the server side. To further streamline data replication between clients, especially replication to the new clients, peer-to-peer protocols have been studied[31].

Both CouchDB and PouchDB are document-based database systems. In this design, documents are equivalent to the row in SQL-like databases, but provide great flexibility by not enforcing any schema. This is required to facilitate flexible data frame design[20] and tolerate missing or corrupt data due to bit-flips. The satellite data frames database uses key-value pairs defined as JavaScript objects, e.g. JSON.

As stated earlier, PouchDB can serve complete HTML5 web pages and run JavaScript applications from its own database. Using this technique, clients can be configured to process the data, store in the client-specific PouchDB database, and replicate it back to the central server. The central server can perform a MapReduce operation[32] on all client-processed data, eliminating any inconsistencies, before adding to the main storage. MapReduce is a well-studied and understood parallelized data processing approach and literature provides numerous successful applications as well challenges faced by using this method[33][34]. The method is designed and therefore well suited for recurring queries and data preprocessing: batch-processing[35]. It should be stressed that setting up mappers and reducers for an on-demand one-time query is convoluted and performance is slower compared to the classical SQL querying (provided that the data fits into a single machine).

Any architecture relying heavily on client-side data generation requires an extensive security analysis that is beyond the scope of this paper. For the sake of argument, running a JavaScript based database and processing scripts makes reverse-engineering of the code and the authentication methods trivial. Additionally, the running system can be modified by a malicious user in runtime. Losing the edge on the data ingestion means that a single malicious machine would be able to 1) ingest a tremendous amount of data into the framework saturating the central database, 2) inject executables into the frame data, possibly compromising the central database or other client applications via a peer-to-peer connection.

Additional to the security threats, the architecture is a classic example of a vendor-lock-in: shifting to a different database system would require significant efforts reducing the flexibility of the future system development along with adding a set of risk factors[36][37].

### *3.3 Lessons learned*

Merging the challenges faced with the production grade legacy system[38] with the experimental CouchDB-PouchDB partial implementation revealed a list of attention points to be addressed by the final system design and listed in the following sub-sections.

#### *3.3.1 Security*

It is evident that the system security should be considered in the earliest stages of the study. Considering a broader picture entailing data protection and recovery is especially important. As well the quantifying and categorizing users based on data access permissions and data

Copyright © 2017 by the International Astronautical Federation (IAF). All rights reserved.

querying.

### *3.3.2 Client-side data replication and processing*

Client-side data processing introduces many variables to the system and not necessarily pays off in the long run. Various client environments oppose different challenges, in case of PouchDB, the operating system and browser contain security features interfering with operations. As an example, the browser local storage is limited to 5Mb on iOS devices and on Safari (MacOS) requests users to validate the local storage permissions incrementally.

JavaScript has the capability to run in parallel, but executing scripts negatively affects the data replication speed. Additionally, after a browser restart, the earlier replicated data could be invalidated, requiring a complete re-download.

### *3.3.3 Data ingestion*

The analysis of the historical data revealed that radio amateurs are sparsely scattered around the world, with a density peak in proximity to the Delft ground station, resulting in data duplication. A similar result could also happen due to the non optimal planning in a distributed ground station network or due to redundancy in receiving stations. Different network performances (or the temporary unavailability of network connection) could also result in out-of-order data frame ingestion on the database side. Data processing scripts may contain errors, requiring re-computation of the complete datasets.

## *3.4 Transition to Big Data: Lambda architecture*

Looking at previously described systems exposes a common flaw. Telemetry frames received by the telemetry server, are processed and added to the central server, at which point the data distribution to the clients take place. Running both systems revealed that next to machine-tolerance, system should have human-tolerance, as bugs in data processing are frequent [38] and arguably unavoidable. Additionally, the system faces a more general challenge: on the one hand, near-real time data processing is required, while on the other hand, datasets are expected to be consistent and reliable. This correlates back to the CAP theorem [28]: any data storage system can only ensure two out of three characteristics: Consistency, Availability or Partition Tolerance. Originally proposed by Marz [29], as a solution to reduce the system complexity by "the use of mutable state in databases and the use of incremental algorithms to update that state", provides a solution to the problem. A common implementation is an architecture consisting split into two parts, one for incremental state update: speed layer and one containing immutable data used for analysis: batch layer [39]. The system architecture following this approach is commonly known as

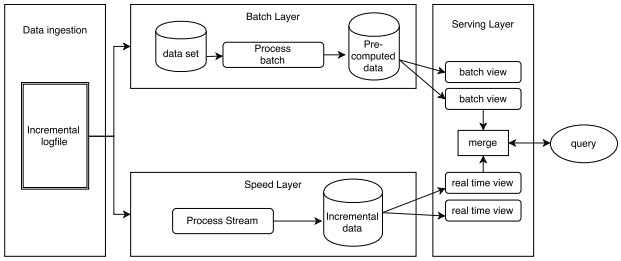


Fig. 5: Lambda Architecture

Lambda architecture. Figure 5 provides a high level functional overview. It should be noted that each node on the diagram represents a server cluster.

### *3.4.1 Data consumption*

As shown in figure 5 data consumption is the data entry point to the system. Architecture does not oppose requirements on the design of this component, however, all incoming data has to be split into two identical streams, one consumed by batch layer, one by speed layer.

### *3.4.2 Batch layer*

The batch layer has two functions, appending new data to the immutable data storage, and computing the batch view. It should be stressed that all received data is stored permanently, preferably without the ability to be modified, preventing data corruption due to human interaction. Once stored, data ought to be processed in batches, eliminating the data inconsistencies such as duplicates and out-of-order frames.

### *3.4.3 Streaming layer*

Running large batch jobs is both time and resources consuming, and is therefore expected to be executed on intervals. The streaming layer, designed to compensate for the data between the batch intervals, depends on the real-time data arriving the system and is therefore completely independent of the immutable storage. An interesting consequence of this is that the stream processing generates own stream views, potentially containing out-of-order and duplicate data.

### *3.4.4 Serving layer*

Serving layer is responsible for running queries on the collection of the streaming and batch views. As previously stated, all historic data is present in the batch view, while all the newly received data can be found in the stream views. The techniques of executing the queries and removing redundant stream view data upon batch completion are not enforced by the architecture and is part of the implementation.

Copyright © 2017 by the International Astronautical Federation (IAF). All rights reserved.

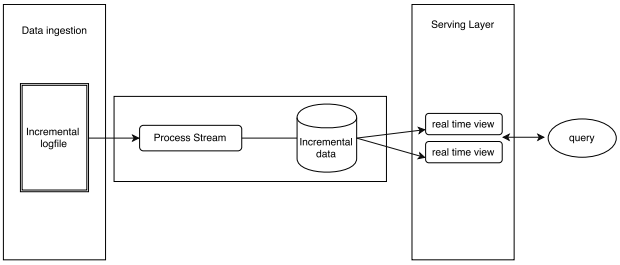


Fig. 6: Kappa Architecture

### *3.5 Transition to Big Data: Kappa architecture*

When introduced, Lambda architecture relied heavily on MapReduce as batch processor and Apache Storm and Apache Flink for stream processing layer. With the maturity of Spark Streaming API, a number of alternatives architectures has been proposed, Kappa architecture as the most popular one. As shown in Figure 6, Kappa is a simplification of Lambda architecture, with the batch processing and stream processing aggregated to a single Spark cluster. Spark batch processing, sharing the code base with Spark Streaming, can be executed on demand to reprocess the complete historic data set. While for the day-to-day operations system would utilize the Spark streaming. This approach simplifies the serving layer, removing the double views.
When working with stream processing, it is important to keep transactions stateless whenever possible. Relying on database reads to validate the data, to remove duplicates or out-of-order frames, is not welcomed. In some cases batch processing will be required to remove or update past event states, what in its turn invalidates earlier streaming views, requiring extra complexity to mitigate the downtime.

## 4. Selected architecture

The data processing system is designed with two main functions in mind: provide satellite monitoring functionality and facilitate data mining. The satellite monitoring entails telemetry data processing and visualization. Data mining entails identifying data sources, building and validating data models iteratively, and potentially embedding data model outcome in the monitoring dashboard. From the earlier proposed architectures, only Lambda and Kappa can be considered as viable options. For data transformation and visualization purposes, Kappa architecture proves as well suited candidate, due to its streamlined and lean approach. However, by oposing data mining requirements, frequent batch jobs become a necessity for building, validating and improving of the data models. Additionally,by regularly recalculating the complete historic data set, ensures data consistency in the serving layer, making data more accessible for querying by satellite operators. The abstraction of the Batch layer opens opportunities to run jobs on separate (cloud compute) clusters, cutting processing time for resource demanding computations without affecting satellite operations. This leads to the selection of Lambda architecture for the project.

Following sections cover high-level decisions and framework selection. The groundwork of available applications is well covered in the literature [40] [41] [42],therefore only high-level description will be provided.

### *4.1 Serving layer*

The serving layer is designed to aggregate and serve data from streaming and batch layers to the client application. This function can be fulfilled in many ways, for example by running a single database system or a query engine on two different database systems. The implementation depends on the user requirements, in scope of this project, users require near-real-time graphical and tabular views of the satellite status (dashboard). Additionally, the user should be able to execute custom, on-demand queries for data analysis and satellite troubleshooting.

Since the satellite telemetry is techincally time-series data, the majority of the existing log data visualization frameworks can be applied out of the box. Kibana and Grafana are two most popular and powerful open-source visualization tools [43]. Grafana is designed with a time-series database on the backend in mind, while Kibana utilizes ElasticSearch. Grafana supports multiple databases following strict time-series schema, while Kibana, only supports ElasticSearch but allowing more flexible schema. Recently published work [44] proves both frameworks comparable on the visualization aspects but requiring further research on graphing capabilities for the actual satellite telemetry. ES enables users to execute queries and calculations within Kibana. While querying time-series database, such as InfluxDB, requires an additional interface to bind to the database API. Providing this functionality to users over the internet increasing overall complexity and requires careful design and implementation.

For the project Kibana and ElasticSearch has been selected, due to its operational simplicity, features and ease of implementation and maintenance.

Utilizing Kibana requires both Streaming as Batch processed data to be stored in ElasticSearch. Aggregating this data requires removal of redundant Streaming Data upon Batch completion. This is not a unique problem [45] and can be resolved using ElasticSearch Curator by assigning retention to the streaming data. Another solution to the problem is to overwrite all data in ElasticSearch on Batch completion, actively removing all Streaming Data.

Copyright © 2017 by the International Astronautical Federation (IAF). All rights reserved.

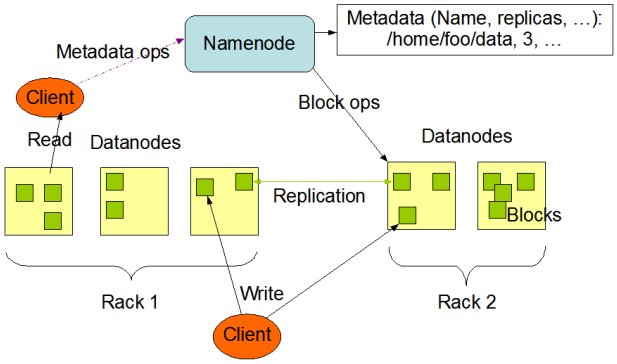


Fig. 7: HDFS Architecture [46]

Further research is required to determine the optimal technique.

### *4.2 Batch layer*

The batch layer is introduced to the Lambda architecture with the purpose of bulk data processing and retention of the immutable data set.

#### *4.2.1 Immutable data*

The storage of the immutable data set requires a system that ensures file integrity while allowing access to multiple (simultaneous) readers. This can be achieved using (distributed) file system or a database system. Hadoop Distributed File System has been selected over databases systems due to

- Tolerance to unstructured data: satellite data can use different formats, from binary blobs to image and video file(streams)
- Scalability: deployed and extended to multiple machines without configuration changes to the client applications
- Analytics: HDFS, as part of Hadoop ecosystem, is universally supported for data processing, includes API's and query engines for custom data processing systems.

Hadoop Distributed File System (HDFS) is a distributed file system designed to provide scalable, fault-tolerant and consistent data storage across large clusters. Ability to store files greater than server capacity, while providing parallel access on multiple machines in a cluster, makes HDFS an attractive choice for Big Data applications.

As shown in the figure 7, HDFS cluster consists of two components: a Name Node (NN) and a set of Data Nodes (DN). HDFS exposes a file system, allowing clients to store files that are automatically broken up into blocks with the predefined size (128Mb by default) and stored redundantly on the Data Nodes. The Name Node acts as a controller, splitting data into blocks and managing blocks location while optimising read and write performance. CRUD as well file open and close operations are handled by Name Node. In this master-slave architecture, only a single Name Node is allowed at all times, making it a weak point of the system. To mediate this issue a secondary Name Node is assigned by Yarn containing a copy of the edit log, reducing the recovery time of the system.

#### *4.2.2 Batch processing: MapReduce, Apache PIG and Apache TEZ*

MapReduce is briefly covered in section 3.2 as part of CouchDB stack. In scope of BigData MapReduce runs natively within Hadoop stack, on top of the HDFS. The general concept is similar, and processing depends on Mappers to transform and Reducers to filter the data. Typically, MR batch job is controlled by Yarn (resources) and systems like Oozie for execution (time allocation). MR follows a master-slave approach , inherited from HDFS, with a single Node Manager running MR Application Master controlling, determining and allocating Map and Reduce tasks over the cluster. It performs an optimization of the job based on resources (CPU, RAM) as well the nodes locally available data to minimize the network bottlenecks. The Application Master is monitored by Yarn and in case of failure, will be relaunched automatically on a different node along with required information to resume the job. The Reduce jobs often, if not always, require aggregation of data from multiple Mappers, likely executed on different nodes, all handled by MR without being programmed in the query. The main drawback of MR is the two-stage process limitation, that can be medicated by chaining multiple MR operations, but decreasing the overall efficiency. Furthermore, the intermediate Mapper results are stored on the nodes hard drive, further degrading the performance [47]. Apache PIG is an infrastructure and a high-level language, PIG Latin, for data analysis programs, evaluating directly on MapReduce and Tez. Apache TEZ is a high-performance MapReduce alternative that relies on complex directed-acyclic-graphs (DAG) and Hadoop Yarn [48].

MapReduce, being part of the Hadoop ecosystem, is added automatically to the data analytics toolset by selecting HDFS as persistent storage. Due to limited performance of MR and processing limitation of PIG, both systems will not be utilized for batch processing use. However, PIG in conjunction with TEZ serves purposes for

Copyright © 2017 by the International Astronautical Federation (IAF). All rights reserved.

data analytics and troubleshooting of the system.

#### *4.2.3 Data analytics: Apache Hive and HBase*

While Apache PIG is designed with scripting in mind, other abstractions have been developed to emulate a SQL database. Apache Hive is an analytics querying framework within Hadoop ecosystem. By design, Hive is optimized for analytics: online analytical processing (OLAP). In short, Hive provides a SQL-like interface (HiveQL) to access the data stored in HDFS file while only enforcing a schema on read. It should be noted that Hive does not provide record-level updates, inserts or deletes.
Apache HBase, a No-SQL alternative, is designed for online transaction processing (OLTP), similar to Google Big Table. Data records, stored in HDFS, are parsed to column and column families to mitigate missing data.
Nor Hive or HBase is required for batch processing. Hive is part of the Hadoop ecosystem, and will be available for the data analytics.

#### *4.2.4 Apache Spark*

Apache Spark is a popular framework used for big data analytics. Spark is deployed as a cluster application and can be monitored by YARN. In contrast to the two stage MapReduce, Spark executes multi-stage jobs in-memory, drastically improving the overall system performance. The core of Spark relies on the resilient distributed datasets (RDD) [49], abstraction for the partitioned collection of records. This ensures fault-tolerance and an ability to recompute damaged partition with data distributed over the cluster. The fault-tolerance is achieved by keeping all RDD's read-only, ensuring that every transformation creates a new RDD, making each RDD traceable and re-computable. The key to performance is DAG and the policy of transforming the RDDs only when directly dependent downstream RDDs are requested: lazy transformation.
The Spark stack consists of Spark Core, Spark Streaming, Spark SQL, MLlib and GraphX. Spark Core exposes high-level RDD and dataset API for batch data manipulation. Spark API supports a number of programming languages, Scala and Python being the most popular. Spark SQL exposes a SQL-like language for interaction with RDD, utilizing the structured data API. Spark MLlib is a module for machine learning utilizing the RDD abstractions. MLlib provides classification functionality, for example, K-Means clustering, providing the necessary frameworks for basic anomaly detection.
Spark is one of the most versatile batch processing tool available. This is required since the satellite data frame format [20] requires additional processing or decoding tools such as AVRO. Additionally, Spark Streaming allows the majority of code (structured data API) to be reused for both stream and batch processing.

### *4.3 Speed Layer*

The speed layer requires fast processing while ensuring fault-tolerance and reliability to deliver data timely to the connected clients. At the time of writing, three distributed stream processing framework dominate the scene: Apache Storm, Apache Spark and Apache Flink. The frameworks are well studied and number of publications are made on the trade-off and benchmarks. [50] [51] For the purpose of this project Apache Spark has been selected. The core of data frame processing is identical for Speed and Batch layers; utilizing the same data processing framework, allows reuse of the code as well cluster, reducing the overhead.

### *4.4 Ingestion layer*

Ingestion layer ought to provide a secured interface for the client applications to communicate with, undependable from server implementation and frameworks used. The API design is out of the paper scope and will be ignored for the discussion. The ingestion layer should be horizontally scalable and provide (temporary) data storage in case of immutable storage malfunctions (resilience).

#### *4.4.1 API + HDFS*

Hadoop Distributed File System (HDFS) exposes a programming interface that can be easily connected to the API used for client communication (data ingest), enabling direct data consumption by the cluster. This solution, however, requires high availability HDFS deployment to cover for any malfunctions, and a system to feed the streaming data to the Speed Layer.

#### *4.4.2 Kafka*

Defacto framework used for the ingestion layer in lambda architecture is Apache Kafka. Designed as a system to deliver high volume event data to subscribers, Kafka utilizes a write-ahead commit log on persistent storage and provides a pull-based messaging abstraction to allow both real-time subscribers such as online services and offline subscribers such as Hadoop and data warehouse to read these messages at arbitrary pace. [52].

The stream of records, published by API, are categorised in topics. Topics are used to define data pipelines and are consumed by subscribers: Speed layer and Batch layers. In case of clustered deployment, topics are build up from partitions, collectively called log. As shown in the Figure 8, each partition is an immutable sequence of received data. Offset, the unique id of each record is used to keep track of the last retrieved record per subscriber.

Copyright © 2017 by the International Astronautical Federation (IAF). All rights reserved.

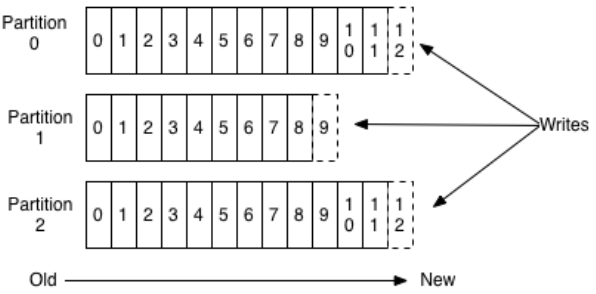


Fig. 8: Kafka Log Anatomy [53]

This enables topic subscribers to consume data at different rates. To keep track of offset and subscribers Kafka utilizes Apache Zookeeper. Additional to the messaging broker, Kafka contains Connect framework, an extensive set of ready-to-use Sink and Source connectors for integrating with the majority of existing databases and data providers. The framework is optimal for data migration from legacy system and load testing of the complete system implementation.
The choice for Kafka is made due to the following considerations:

- Horizontally scalable
- Ability to serve multiple data consumers at different rates
- Resilient data log, redundancy for temporarily HDFS system failures.
- Data delivery guarantee

### *4.5 Final Architecture*

Selection process briefly outlined by previous sections, leads to the architecture shown by the Figure 9.
Satellite telemetry data submit through client application and API, is appended to commit log of Apache Kafka. Log serves as a temporarily data storage, until it is consumed by Spark Streaming and inserted to HDFS file system by Apache Connect HDFS Sink. Spark batch processing is executed on regular intervals, result of which overwrites all entries in the ElasticSearch system. Apache Spark Streaming is executed in micro batches with sub minute intervals. Processed data is appended to ElasticSearch with retention period. Kibana is configured to consume ElasticSearch data.

The architecture ensures interoperability with different components, for example an additional No-SQL database for specific customer needs, withouh major code overhaul. The batch layer, with aid of Spark can be used for a wide range of tasks, from model fitting (K-means) to full fledged machine learning.

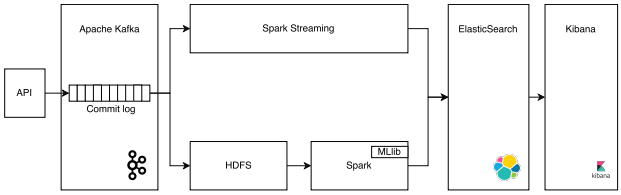


Fig. 9: Proposed Architecture including frameworks

## 5. Future Work

One of the main long-term goals of the work presented here (and still under development) is the use of satellite data (including payload data as well as on-board telemetry) for data mining and autonomous operations. Data mining is defined as the analysis of large amounts of data to extract further information from it: a clear example could be the analysis of performances indicators to predict system maintenance[54]. Spacecrafts could, for example, benefit from a predictive model calibrated around selected telemetry parameters, to predict eventual faults and implement corrective strategies in a completely autonomous way. This latter approach can be very fascinating, especially for deep-space probes that experience long communication delays and requires complex autonomous operations, but it would have to be ported to the satellite hardware.

Using historic telemetry data coming from space probes has also been exploited recently by the Mars Power Challenge[55], where the best modeling techniques were compared to predict 1 year worth of telemetry on the Mars Express probe based on 3 years worth of data. Possible artificial intelligence applications in this case would have to be integrated either on the satellite hardware or in the ground station infrastructure, leading to further implementation work. In our case, a future implementation of data mining algorithms will be simple to add because we already relied on standard applications used in the artificial intelligence / data mining field to realize the database and the data distribution system since this could be implemented by the processing layer already present. This approach will allow us to perform further research even during normal mission operations with the clear goal of creating an automated system to handle common anomalies.

## 6. Conclusions

In this paper we looked at the current trends in space missions, especially looking at nano-satellites, and fo-

Copyright © 2017 by the International Astronautical Federation (IAF). All rights reserved.

cused on swarms and constellations. From these missions, we looked at the required ground segment to fulfill the mission requirements of handling up to thousands of passes per day. This requires the development of a distributed ground station system capable of scaling in a simple way. We presented an architecture to achieve such goals based on industry standard applications in the domain of data analytics and mining. We also presented some preliminary results on the implementation of such a system to clearly show the advantages of the selected architecture.

We also highlighted possible future developments making use of the described infrastructure to perform data mining and possibly autonomous operations by adding a data mining / artificial intelligence application to the existing distributed database. This new concept could provide huge benefits to big constellations by heavily reducing the operators work.

## References


[1] "2017 Spaceworks Nano/Microsatellite Market Forecast." http://spaceworksforecast.com. (accessed 01.09.2017).

[2] "Gunter's space page." http://space.skyrocket.de. (accessed 02.09.2017).

[3] N. Crisp, K. Smith, and P. Hollingsworth, "Launch and deployment of distributed small satellite systems," *Acta Astronautica*, vol. 114, pp. 65 – 78, 2015.

[4] R. Sandau, *Implications of new trends in small satellite development*, pp. 296–312. Vienna: Springer Vienna, 2011.

[5] J. Esper, P. V. Panetta, M. Ryschkewitsch, W. Wiscombe, and S. Neeck, "Nasa-gsfc nano-satellite technology for earth science missions," *Acta Astronautica*, vol. 46, no. 2, pp. 287 – 296, 2000. 2nd IAA International Symposium on Small Satellites for Earth Observation.

[6] L. Dyrud, S. Slagowski, J. Fentzke, W. Wiscombe, B. Gunter, K. Cahoy, G. Bust, A. Rogers, B. Erlandson, L. Paxton, and S. Arnold, "Small-sat science constellations: why and how," in *Proceedings of the 27th Annual AIAA/USU Conference on Small Satellites*, (Lugan, UT), American Institute of Aeronautics and Astronautics (AIAA), 8 2013.

[7] D. J. Barnhart, T. Vladimirova, A. M. Baker, and M. N. Sweeting, "A low-cost femtosatellite to enable distributed space missions," *Acta Astronautica*, vol. 64, no. 11, pp. 1123 – 1143, 2009.

[8] R. Sandau, K. Brie, and M. DErrico, "Small satellites for global coverage: Potential and limits," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 65, no. 6, pp. 492 – 504, 2010. ISPRS Centenary Celebration Issue.

[9] W. Saylor, K. Smaagard, N. Nordby, and D. Barnhart, "New scientific capabilities enabled by autonomous constellations of smallsats," in *Proceedings of the 21th Annual AIAA/USU Conference on Small Satellites*, (Lugan, UT), American Institute of Aeronautics and Astronautics (AIAA), 8 2007.

[10] D. J. Barnhart, T. Vladimirova, and M. N. Sweeting, "Very-small-satellite design for distributed space missions," *Journal of Spacecraft and Rockets*, vol. 44, no. 6, pp. 1294 – 1306, 2007.

[11] "Planet web page." https://www.planet.com. (accessed 04.09.2017).

[12] K. Colton and B. Klofas, "Supporting the flock: Building a ground station network for authonomy and reliability," in *Proceedings of the 30th Annual AIAA/USU Conference on Small Satellites*, (Lugan, UT), American Institute of Aeronautics and Astronautics (AIAA), 8 2016.

[13] "Spaceflight web page." http://spaceflight.com. (accessed 04.09.2017).

[14] B. Klofas, "Planet labs ground station network." 13th Annual CubeSat Developers Workshop, 4 2016.

[15] C. Venturini and T. McVittie, "Current and future ground systems for cubesats working group," in *Ground Systems Architecture Workshop*, The Aerospace Corporation, 3 2014.

[16] E. F. Moreira, A. Ceballos, C. Estvez, , J. C. Gil, S. Kang, J. Guiney, , and V. Ivatury, "Architecting oneweb's massive satellite constellation ground system," in *Ground Systems Architecture Workshop*, The Aerospace Corporation, 3 2017.

[17] K. Casey, W. Al-Masyabi, and M. Nagengast, "A visit to 2037," in *Ground Systems Architecture Workshop*, The Aerospace Corporation, 3 2017.

[18] "Delfi Space: TU Delft Small Satellite Program." http://www.delfispace.nl/operations/radio-amateurs. (accessed 02.09.2017).

[19] M. Kuiper, "DUDe Telemetry Client Software Design ," tech. rep., Delft University of Technology, 2013.

Copyright © 2017 by the International Astronautical Federation (IAF). All rights reserved.

[20] R. Schoemaker, "Robust and flexible command & data handling on board the delffi formation flying mission," Master's thesis, Delft University of Technology, 2014.

[21] G. van Craen, "Design of the telemetry server," Master's thesis, Delft University of Technology, 2011.

[22] J. Lewis and M. Fowler, "Microservices: a definition of this new architectural term," 2014.

[23] D. Haney and K. S. Madsen, "Load-balancing for mysql," *Kobenhavns Universitet*, 2003.

[24] M. Villamizar, O. Garcés, H. Castro, M. Verano, L. Salamanca, R. Casallas, and S. Gil, "Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud," in *Computing Colombian Conference (10CCC), 2015 10th*, pp. 583–590, IEEE, 2015.

[25] M. Villamizar, O. Garces, L. Ochoa, H. Castro, L. Salamanca, M. Verano, R. Casallas, S. Gil, C. Valencia, A. Zambrano, *et al.*, "Infrastructure cost comparison of running web applications in the cloud using aws lambda and monolithic and microservice architectures," in *Cluster, Cloud and Grid Computing (CCGrid), 2016 16th IEEE/ACM International Symposium on*, pp. 179–182, IEEE, 2016.

[26] J. Justin and J. Jude, "Go offline," in *Learn Ionic 2*, pp. 79–97, Springer, 2017.

[27] "Pouchdb: The database that syncs." `https://pouchdb.com`. (accessed 02.09.2017).

[28] S. Gilbert and N. Lynch, "Perspectives on the cap theorem," *Computer*, vol. 45, no. 2, pp. 30–36, 2012.

[29] N. Marz, "How to beat the cap theorem," *nathanmarz. com*, 2011.

[30] "Couchdb: the definitive guide." `http://guide.couchdb.org/draft/consistency.html`. (accessed 02.09.2017).

[31] R. Leeds, "Chrome to chrome pouchdb." CouchDB Conf Berlin, 2013.

[32] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[33] S. N. Khezr and N. J. Navimipour, "Mapreduce and its applications, challenges, and architecture: a comprehensive review and directions for future research," *Journal of Grid Computing*, pp. 1–27, 2017.

[34] S. A. Thanekar, K. Subrahmanyam, and A. Bagwan, "A study on mapreduce: Challenges and trends," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 4, no. 1, pp. 176–183, 2016.

[35] R. Singh and P. J. Kaur, "Analyzing performance of apache tez and mapreduce with hadoop multinode cluster on amazon cloud," *Journal of Big Data*, vol. 3, no. 1, p. 19, 2016.

[36] D. S. Kusumo, M. Staples, L. Zhu, H. Zhang, and R. Jeffery, "Risks of off-the-shelf-based software acquisition and development: A systematic mapping study and a survey," 2012.

[37] A. Shvets, *Design Patterns Explained Simply*. Source Making, 2017.

[38] S. van Kuijk, "Delfi-n3xt forensics: A hybrid methodology," Master's thesis, Delft University of Technology, 2016.

[39] N. Marz and J. Warren, "Big data: principles and best practices of scalable real-time data systems," 2013.

[40] V. Chavan and R. N. Phursule, "Survey paper on big data," *Int. J. Comput. Sci. Inf. Technol*, vol. 5, no. 6, pp. 7932–7939, 2014.

[41] D. Singh and C. K. Reddy, "A survey on platforms for big data analytics," *Journal of Big Data*, vol. 2, no. 1, p. 8, 2015.

[42] V. B. Bobade, "Survey paper on big data and hadoop," *Int. Res. J. Eng. Technol*, vol. 3, no. 1, pp. 861–863, 2016.

[43] A. Yigal, "Grafana vs. kibana: The key differences to know." `https://logz.io/blog/grafana-vs-kibana`. Accessed: 2017-09-02.

[44] I. Nurgaliev, E. Karavakis, and A. Aimar, "Kibana, grafana and zeppelin on monitoring data," Aug. 2016.

[45] P. Kleindienst, "Building a real-world logging infrastructure with logstash, elasticsearch and kibana,"

[46] "Hdfs architecture guide." `https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html`.

[47] P. Kannan, "Beyond hadoop mapreduce apache tez and apache spark," *San Jose State University. URL: http://www. sjsu. edu/people/robert. chun/courses/CS259Fall2013/s3/F. pdf*.

Copyright © 2017 by the International Astronautical Federation (IAF). All rights reserved.

[48] B. Saha, H. Shah, S. Seth, G. Vijayaraghavan, A. Murthy, and C. Curino, "Apache tez: A unifying framework for modeling and building data processing applications," in *Proceedings of the 2015 ACM SIGMOD international conference on Management of Data*, pp. 1357–1369, ACM, 2015.

[49] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pp. 2–2, USENIX Association, 2012.

[50] A. Shukla and Y. Simmhan, "Benchmarking distributed stream processing platforms for iot applications," in *Technology Conference on Performance Evaluation and Benchmarking*, pp. 90–106, Springer, 2016.

[51] S. Chintapalli, D. Dagit, B. Evans, R. Farivar, T. Graves, M. Holderbaugh, Z. Liu, K. Nusbaum, K. Patil, B. J. Peng, *et al.*, "Benchmarking streaming computation engines: Storm, flink and spark streaming," in *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International*, pp. 1789–1792, IEEE, 2016.

[52] G. Wang, J. Koshy, S. Subramanian, K. Paramasivam, M. Zadeh, N. Narkhede, J. Rao, J. Kreps, and J. Stein, "Building a replicated logging system with apache kafka," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1654–1655, 2015.

[53] "Kafka documentation." `https://kafka.apache.org/documentation/`.

[54] P. Bastos, I. Lopes, and L. Pires, "A maintenance prediction system using data mining techniques," in *World Congress on Engineering 2012*, vol. 3, pp. 1448–1453, International Association of Engineers, 2012.

[55] "Mars express power challenge." `https://kelvins.esa.int/mars-express-power-challenge`. (accessed 01.09.2017).