Distributed Stream Processing

Linh B. Ngo

MOTIVATION

- Faster communication infrastructures (wired/wireless/cellular)
- Widespread usage of smartphones
  - User-created contents
- Internet of Things
  - Pervasive embedded/wearable sensors

MOTIVATION

- Massive streams of data from multiple sources
  - Volume
  - Velocity
- Distributed Stream Processing
  - Variety
  - Veracity
  - Value

BATCH VS STREAM

- Batch
- Micro-batch
- Stream

	Batch	Stream
Data Unit for Computation	Multiple	Single
Latency Demand	High	Low
Data Persistency	Persistent	Non-persistent
Incremental Result	No	Yes
Order of arrival awareness	No	Yes
Distributed	Yes	Yes

DATABASE VS DATA STREAM

	Databases	Data Streams
Data access	Random	Sequential
Number of passes	Multiple	Single
Processing time	Unlimited	Restricted
Available memory	Unlimited	Fixed
Result	Accurate	Approximate
Distributed	YES ~~No~~	Yes

EXAMPLE

$$A = \frac{x_0 + x_1 + \cdots + x_{n-1}}{n}$$

Time	t_0	t_1	t_2	…	t_{n-1}
Value	x_0	x_1	x_2	…	x_{n-1}
Batch Sum	…	…	…	…	A
Stream Sum	A_0	A_1	A_2	…	A

$$S = S_{n-2} + x_{n-1}$$
$$c = c_{n-2} + 1$$
$$A = S/c$$

STREAM PROCESSING MODEL

- Stream is a sequence of unbounded tuples of the form $(a_0, a_1, \dots, a_{n-1}, t)$ generated continuously in time, with a_i denotes an attribute and t denotes the time.
- A processing unit in a stream processing engine (SPE) is called a processing element (PE)
- SPE creates a logical network of Pes connected in a directed acyclic graph (DAG)
- Similar to Apache Tez model

CODE EXAMPLE

- Get one node from Palmetto (minimum two cores)
- Open two terminals to this node
  - Stream generator: netcat
  - Stream processor: Spark Streaming
- On terminal for stream generator
  - `nc -lk 9999`

CODE EXAMPLE

- On terminal for stream processor: stateless count
  - `cd $SPARK_HOME`
  - `vim examples/src/main/python/streaming/network_wordcount.py`
    - Add `sc.setLogLevel("ERROR")`

    under `sc = SparkContext(appName="PythonStreamingNetworkWordCount")`
    - Save and quit
  - `./bin/spark-submit examples/src/main/python/streaming/network_wordcount.py localhost 9999`

CODE EXAMPLE

- On terminal for stream processor: stateful count
  - `cd $SPARK_HOME`
  - `vim examples/src/main/python/streaming/stateful_network_wordcount.py`
    - Add `sc.setLogLevel("ERROR")`

    under `sc = SparkContext(appName="PythonStreamingStatefulNetworkWordCount")`
    - Save and quit
  - `./bin/spark-submit examples/src/main/python/streaming/stateful_network_wordcount.py localhost 9999`

SPE REQUIREMENT

- Data Mobility
  - How data moves among PEs
  - Key to low latency
- High Availability and Data Processing Guarantees
  - Failure recovery
  - Exactly once, at least once, or no guarantee
- Data Partitioning
  - Parallel processing for large data
- Data Querying
  - Ability to querying stream networks

SPE REQUIREMENT

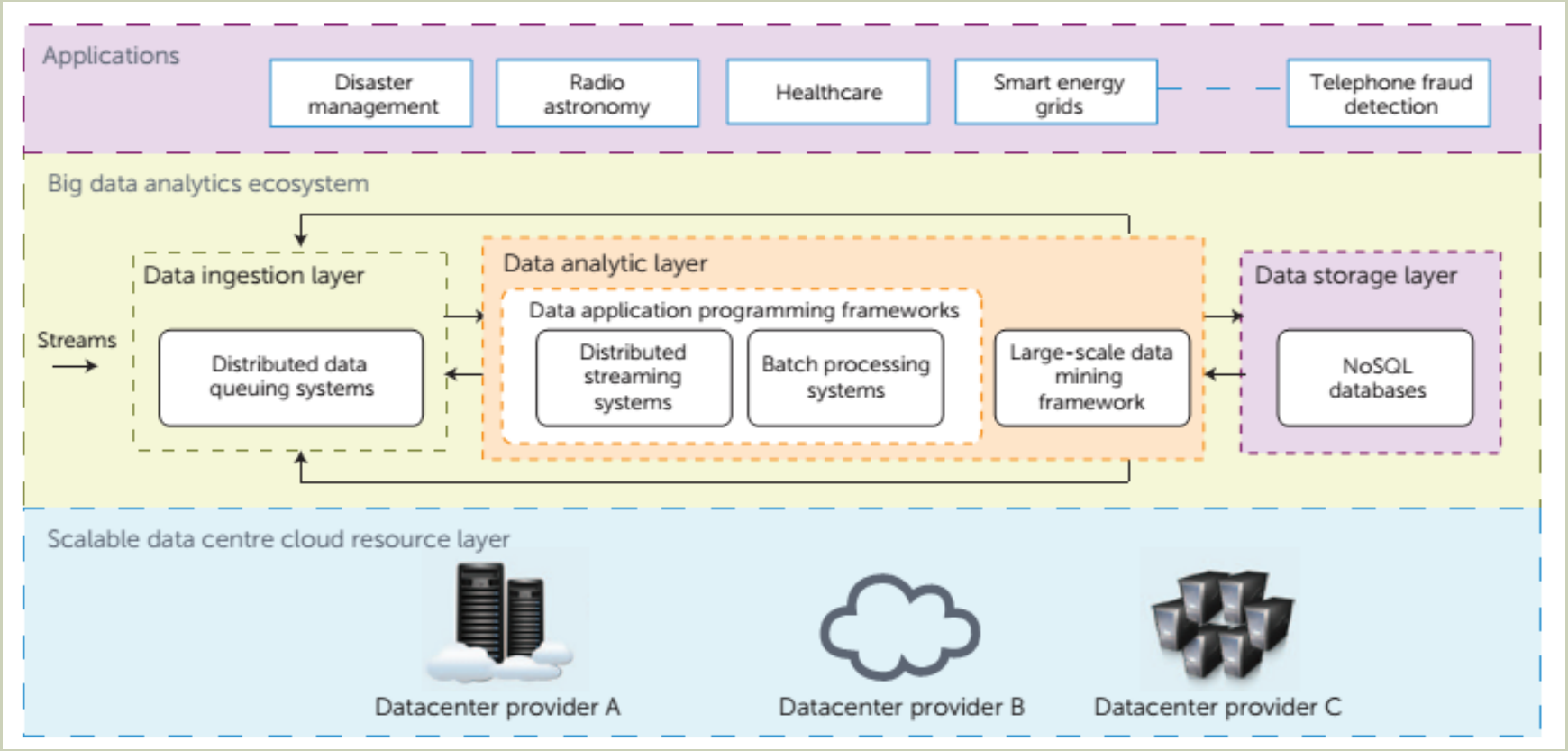
- Deterministic or Non-deterministic Processing
  - Relevant toward failure recovery (think data lineage)
- Data Storage
  - Ability to store and process historical data
- Handling Stream Imperfects
  - Delayed
  - Out of order
  - Duplicated
  - Lost

DISTRIBUTED SPE

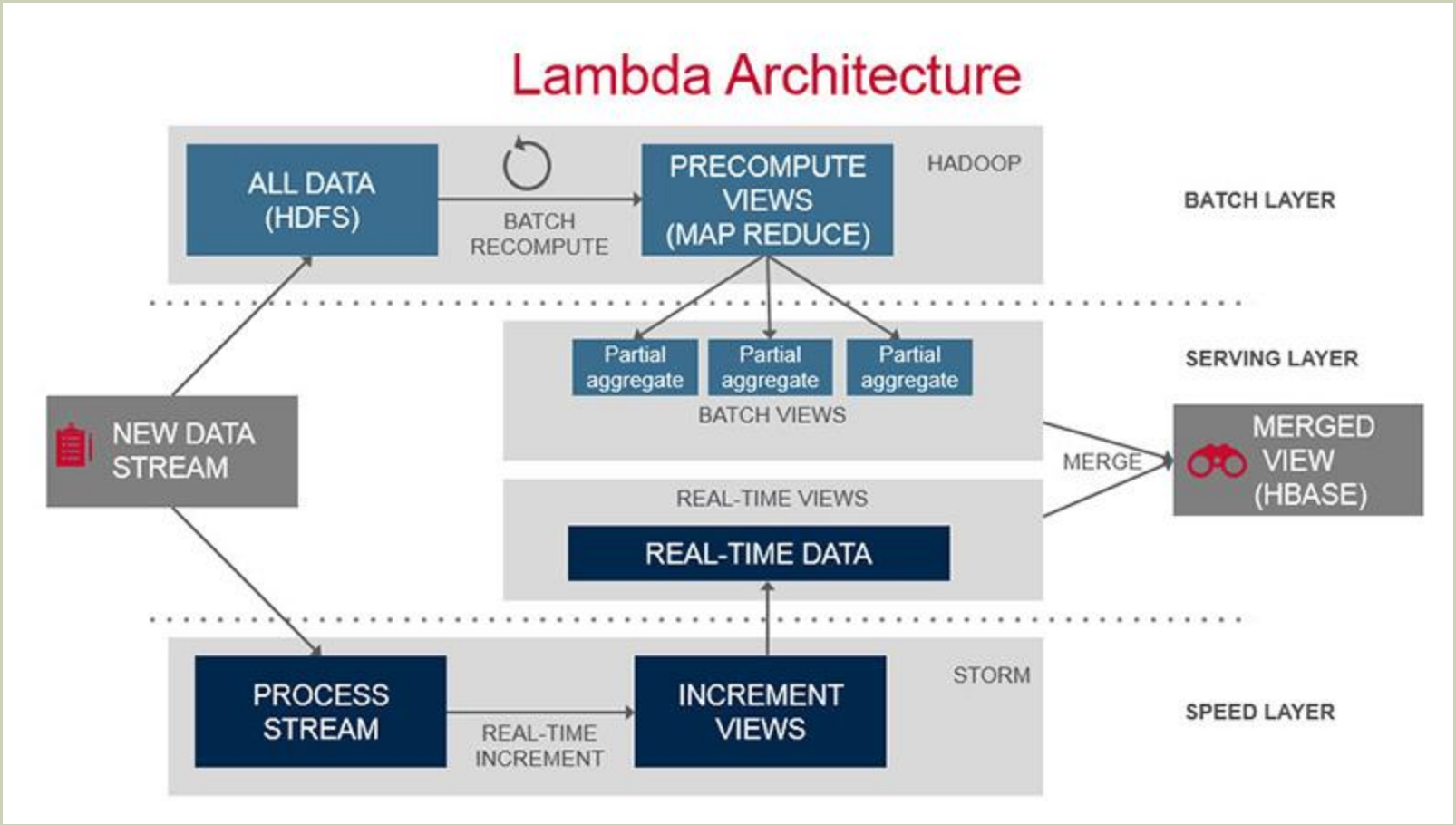
- Aurora
- Borealis
- Apache Storm
- Apache S4
- Apache Samza
- Apache Spark Streaming
- Apache Flink

	Data Mobility	HA & Data Process Guarantees	Data Partition and Scaling	Data Querying	Deterministic vs Non-deterministic	Data Storage	Stream Imperfection Handling
Aurora	Push	Rollback recovery with upstream back up	None	SQL-based	Mostly deterministic	Yes	No
Borealis	Push	Rollback recovery, active back up recovery	Automated by system	SQL-based	deterministic	Yes	Yes
Storm	Pull, non-blocking	Rollback recovery with upstream back up At least once	User configuration	Trident	N/A	No	User
S4	Push	Gap recovery	KV pairs	No	N/A	No	No
Samza	Pull from broker	Rollback recovery SPF at broker	Topic partitioning	No	N/A	Yes (broker)	Yes (broker)
Spark Streaming	Push/Pull	Write-ahead-log for rollback recovery Exactly once	Automated by system	SQL-based	deterministic	Yes	User
Flink	Push/Pull	Light weight snapshot recovery Exactly/At least once	Automated by system	On-going	deterministic	Yes	User

DATA STREAMING INFRASTRUCTURE



LAMBDA ARCHITECTURE



REFERENCES

- Ranjan, Rajiv. "Streaming Big Data Processing in Datacenter Clouds." *Cloud Computing, IEEE* 1.1 (2014): 78-83.
- Gaber, Mohamed Medhat, et al. "Data stream mining in ubiquitous environments: state-of-the-art and current directions." *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 4.2 (2014): 116-138.
- Kamburugamuve, Supun, et al. *Survey of distributed stream processing for large stream sources*. Technical report. 2013. Available at http://grids. ucs. indiana. edu/ptliupages/publications/survey_stream_proc essing. pdf, 2013.