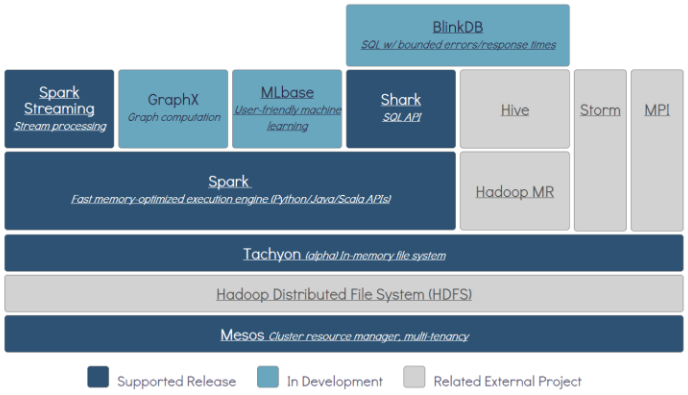
Real-Time Analytical Processing (RTAP) Using the Spark Stack

Jason Dai
jason.dai@intel.com
Intel Software and Services Group

Project Overview

Research & open source projects initiated by AMPLab in UC Berkeley

BDAS: Berkeley Data Analytics Stack
(Ref: https://amplab.cs.berkeley.edu/software/)



Intel closely collaborating with AMPLab & the community on open source development

- Apache incubation since June 2013
- The “most active cluster data processing engine after Hadoop MapReduce”

Intel partnering with several big websites

- Building next-gen big data analytics using the Spark stack
- *Real-time analytical processing* (RTAP)
- E.g., Alibaba , Baidu iQiyi, Youku, etc.

Next-Gen Big Data Analytics

Volume

- Massive scale & exponential growth

Variety

- Multi-structured, diverse sources & inconsistent schemas

Value

- Simple (SQL) – descriptive analytics
- Complex (non-SQL) – predictive analytics

Velocity

- Interactive – the speed of thought
- Streaming/online – drinking from the firehose

Next-Gen Big Data Analytics

Volume

- Massive scale & exponential growth

Variety

- Multi-structured, diverse sources & inconsistent schemas

Value

- Simple (SQL) – descriptive analytics
- Complex (non-SQL) – predictive analytics

Velocity

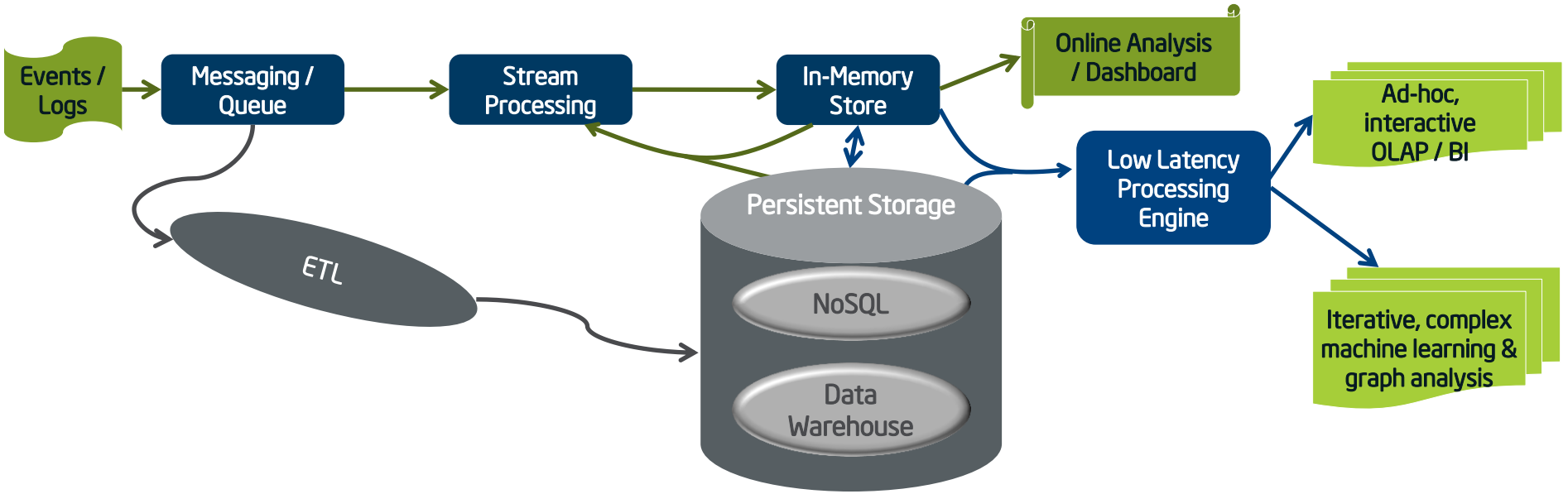
- Interactive – the speed of thought
- Streaming/online – drinking from the firehose

The Spark stack

Real-Time Analytical Processing (RTAP)

A vision for next-gen big data analytics

- Data captured & processed in a (semi) streaming/online fashion
- Real-time & history data combined and mined interactively and/or iteratively
  - Complex OLAP / BI in interactive fashion
  - Iterative, complex machine learning & graph analysis
- Predominantly memory-based computation



Real-World Use Case #1
(Semi) Real-Time Log Aggregation & Analysis

Logs continuously collected & streamed in

- Through queuing/messaging systems

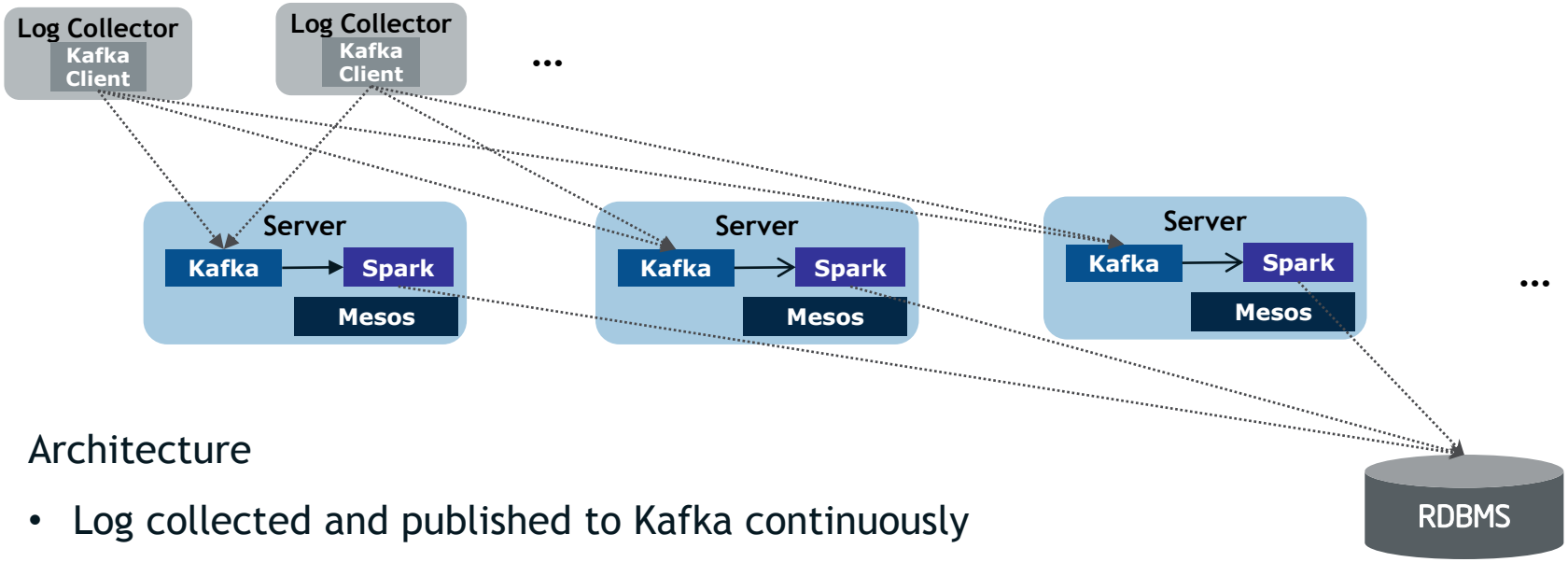
Incoming logs processed in a (semi) streaming fashion

- Aggregations for different time periods, demographics, etc.
- Join logs and history tables when necessary

Aggregation results then consumed in a (semi) streaming fashion

- Monitoring, alerting, etc.

(Semi) Real-Time Log Aggregation: Mini-Batch Jobs



Architecture

- Log collected and published to Kafka continuously
  - Kafka cluster collocated with Spark Cluster
- Independent Spark applications launched at regular interval
  - A series of mini-batch apps running in “fine-grained” mode on Mesos
  - Process newly received data in Kafka (e.g., aggregation by keys) & write results out

In production in the user’s environment today

- 10s of seconds latency

(Semi) Real-Time Log Aggregation: Mini-Batch Jobs

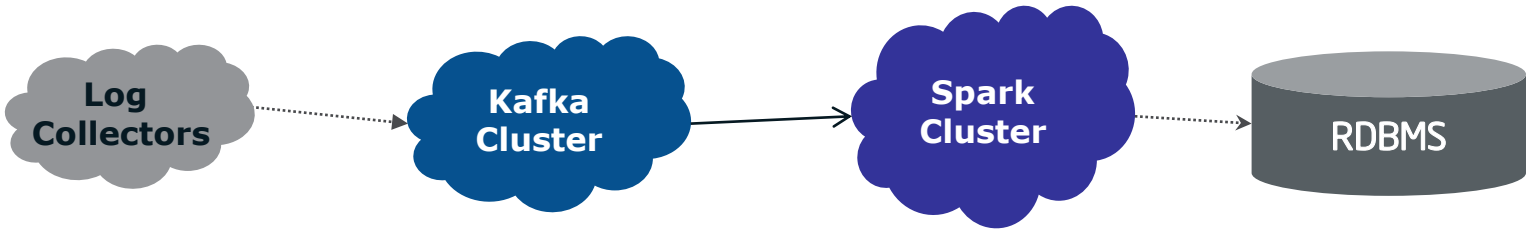
Design decisions

- Kafka and Spark servers collocated
  - A distinct Spark task for each Kafka partition, usually reading from local file system cache
- Individual Spark applications launched periodically
  - Apps logging current (Kafka partition) offsets in ZooKeeper
  - Data of failed batch simply discarded (other strategies possible)
- Spark running in “fine-grained” mode on Mesos
  - Allow the next batch to start even if the current one is late

Performance

- Input (log collection) bottlenecked by network bandwidth (1GbE)
- Processing (log aggregation) bottlenecked by CPUs
  - Easily scaled to several seconds

(Semi) Real-Time Log Aggregation: Spark Streaming



Implications

- Better streaming framework support
  - Complex (e.g., stateful) analysis, fault-tolerance, etc.
- Kafka & Spark not collocated
  - DStream retrieves logs in background (over network) and caches blocks in memory
- Memory tuning to reduce GC is critical
  - `spark.cleaner.ttl` (throughput * spark.cleaner.ttl < spark mem free size)
  - Storage level (`MEMORY_ONLY_SER2`)
- Lowe latency (several seconds)
  - No startup overhead (reusing `SparkContext`)

Real-World Use Case #2
Complex, Interactive OLAP / BI

Significant speedup of ad-hoc, complex OLAP / BI

- Spark/Shark cluster runs alongside Hadoop/Hive clusters
- Directly query on Hadoop/Hive data (interactively)
- No ETL, no storage overhead, etc.

In-memory, real-time queries

- Data of interest loaded into memory

```
create table XYZ tblproperties ("shark.cache" = "true") as select ...
```

- Typically frontended by lightweight UIs
  - Data visualization and exploration (e.g., drill down / up)

Time Series Analysis: Unique Event Occurrence

Computing unique event occurrence across the time range

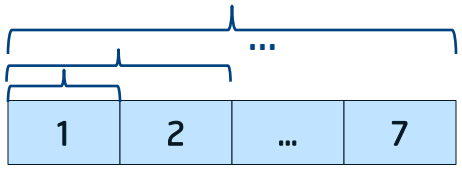
- Input time series

  <TimeStamp, ObjectId, EventId, ...>

  - E.g., watch of a particular video by a specific user
  - E.g., transactions of a particular stock by a specific account
- Output:

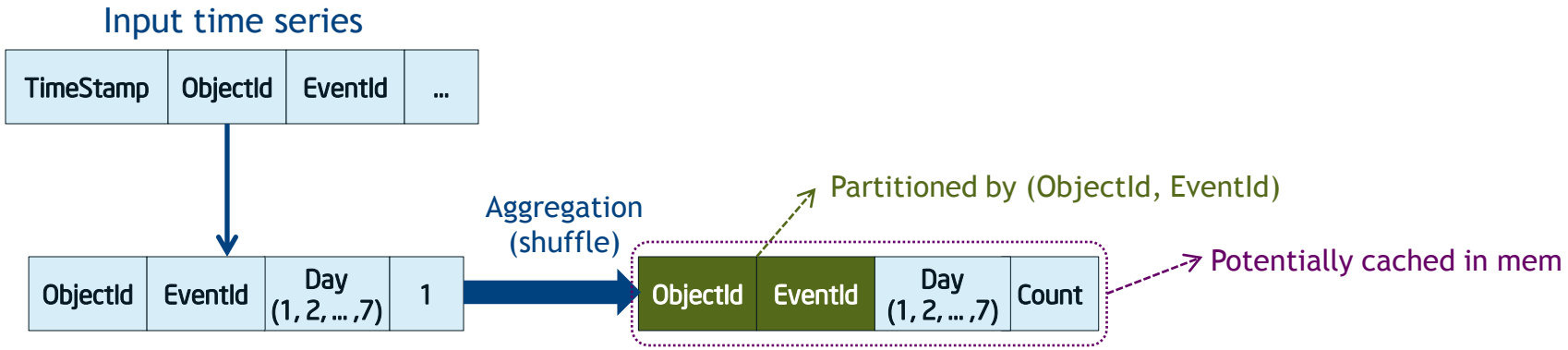
  <ObjectId, TimeRange, Unique Event#, Unique Event(≥2)#, …, Unique Event(≥n)#>

  - E.g., accumulated unique event# for each day in a week (staring from Monday)

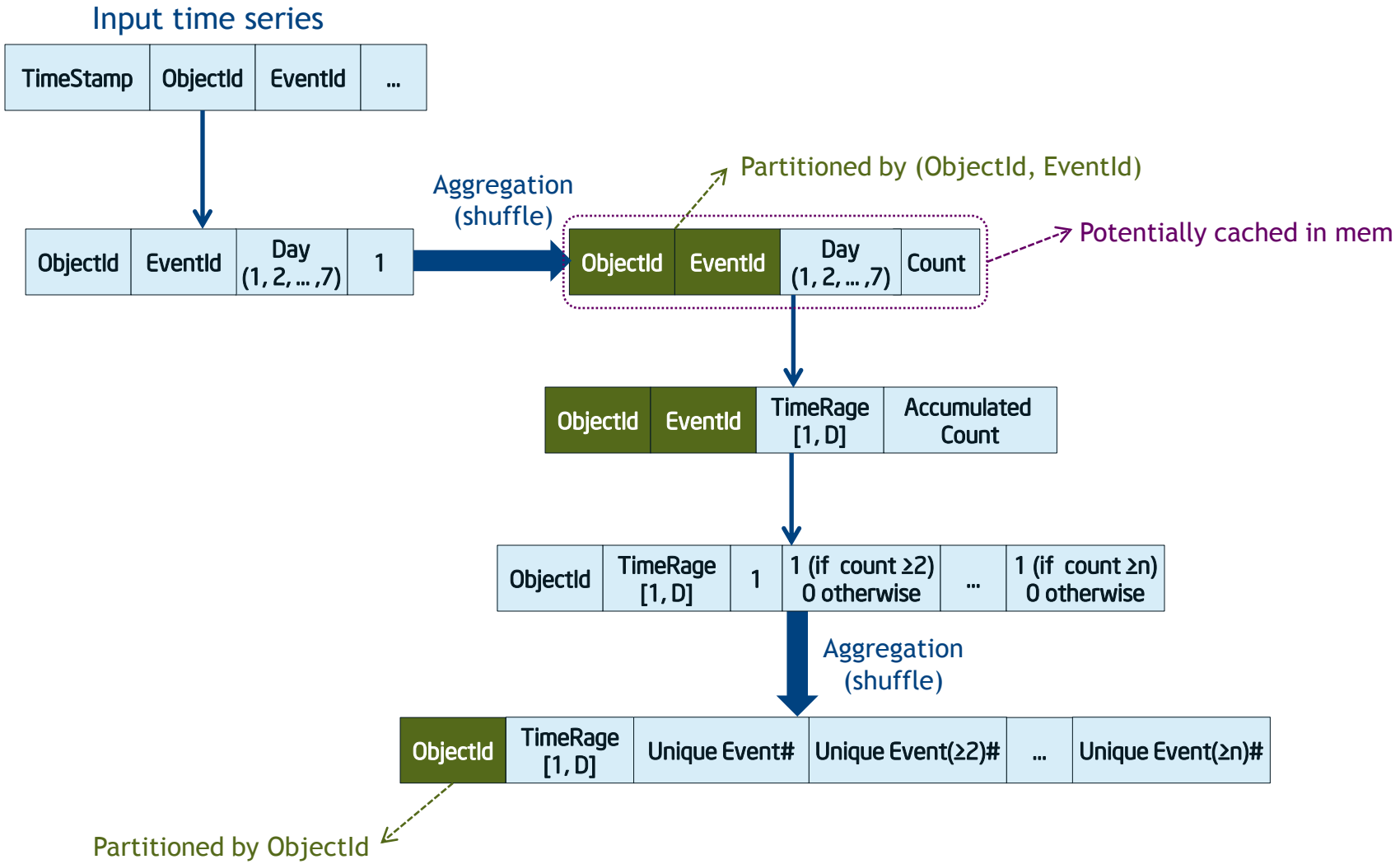


- Implementation
  - 2-level aggregation using Spark
    - Specialized partitioning, general execution graph, in-memory cached data
  - Speedup from 20+ hours to several minutes

Time Series Analysis: Unique Event Occurrence



Time Series Analysis: Unique Event Occurrence



Real-World Use Case #3
Complex Machine Learning & Graph Analysis

Algorithm: complex math operations

- Mostly matrix based
  - Multiplication, factorization, etc.
- Sometime graph-based
  - E.g., sparse matrix

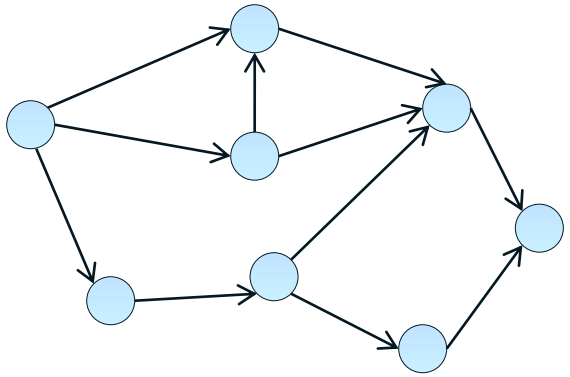
Iterative computations

- Matrix (graph) cached in memory across iterations

Graph Analysis: N-Degree Association

N-degree association in the graph

- Computing associations between two vertices that are *n-hop* away
- E.g., friends of friend



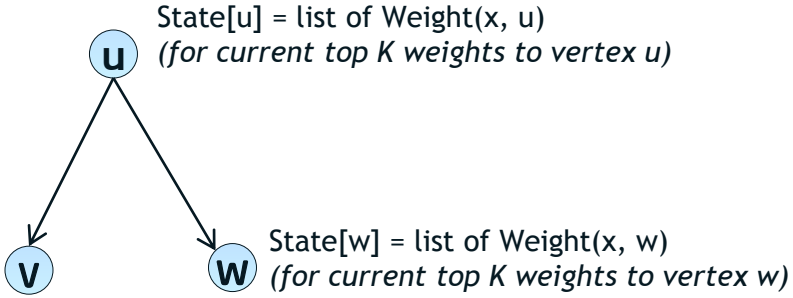
$Weight_1(u, v) = edge(u, v) \quad \in (0, 1)$

$Weight_n(u, v) = \sum_{x \to v} Weight_{n-1(}u, x) * Weight_1(x, v)$

Graph-parallel implementation

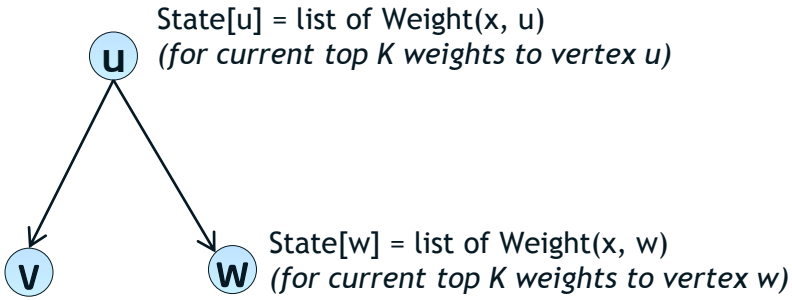
- Bagel (Pregel on Spark) and GraphX
  - Memory optimizations for efficient graph caching critical
- Speedup from 20+ minutes to <2 minutes

Graph Analysis: N-Degree Association

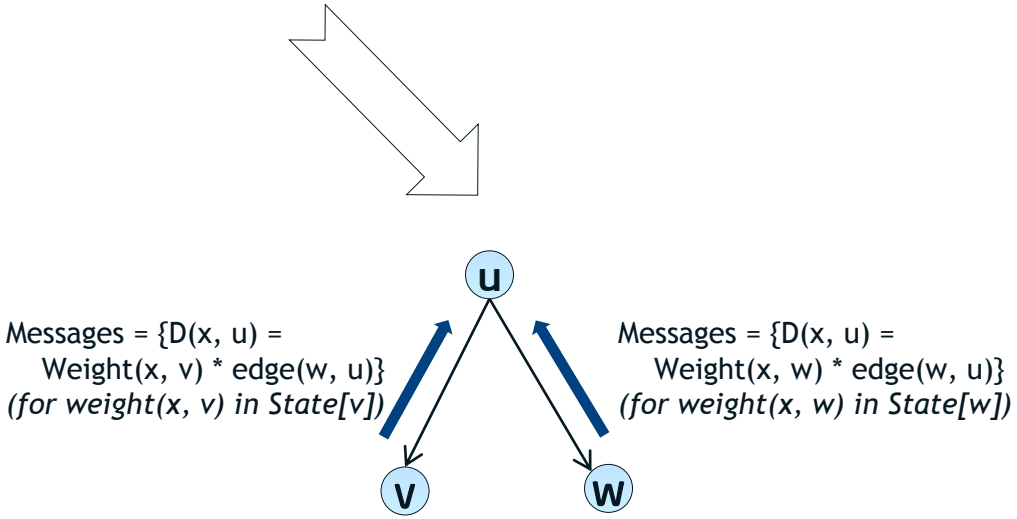


State[v] = list of Weight(x, v)
(for current top K weights to vertex v)

Graph Analysis: N-Degree Association

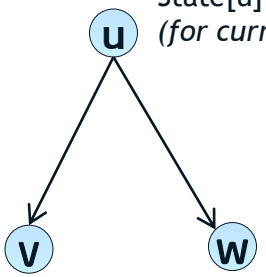


State[v] = list of Weight(x, v)
(for current top K weights to vertex v)



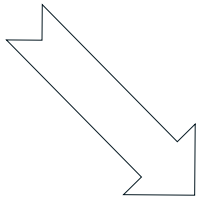
Graph Analysis: N-Degree Association

State[u] = list of Weight(x, u)
(for current top K weights to vertex u)

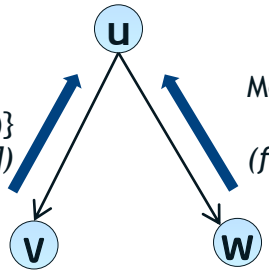


State[w] = list of Weight(x, w)
(for current top K weights to vertex w)

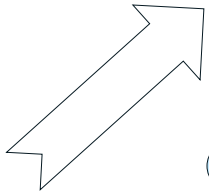
State[v] = list of Weight(x, v)
(for current top K weights to vertex v)



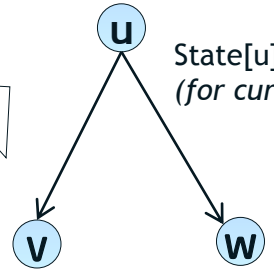
Messages = {D(x, u) =
Weight(x, v) * edge(w, u)}
(for weight(x, v) in State[v])



Messages = {D(x, u) =
Weight(x, w) * edge(w, u)}
(for weight(x, w) in State[w])

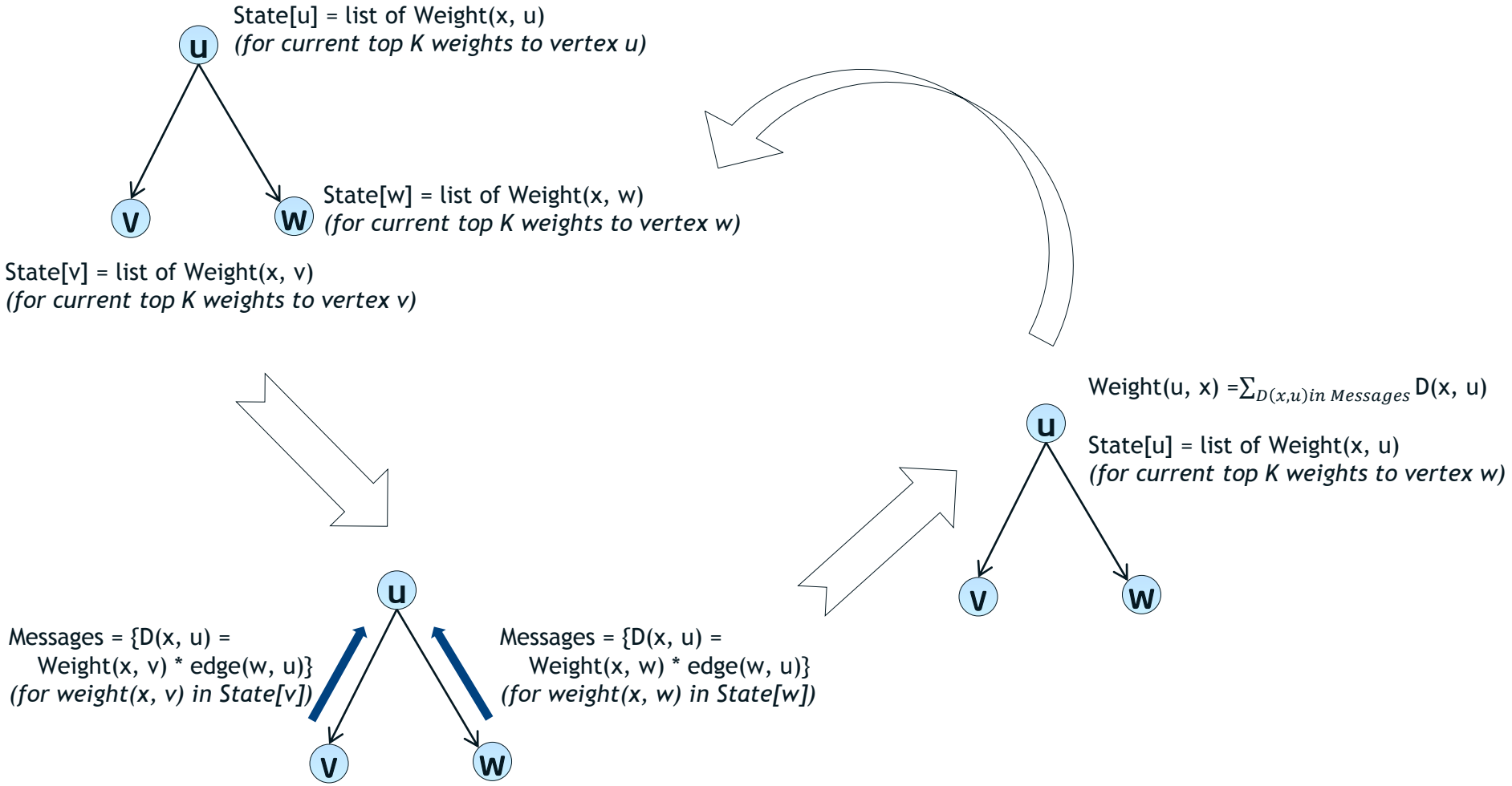


Weight(u, x) = $\sum_{D(x,u) in\ Messages}$ D(x, u)



State[u] = list of Weight(x, u)
(for current top K weights to vertex w)

Graph Analysis: N-Degree Association



Contributions by Intel

Netty based shuffle for Spark

FairScheduler for Spark

Spark job log files

Metrics system for Spark

Configurations system for Spark

Spark shell on YARN

Spark (standalone mode) integration with security Hadoop

Byte code generation for Shark

Co-partitioned join in Shark

. . .

Summary

The Spark stack: lightning-fast analytics over Hadoop data

- Active communities and early adopters evolving
  - Apache incubator project
- A growing stack: SQL, streaming, graph-parallel, machine learning, …

Work with us on next-gen big data analytics using the Spark stack

- Interactive and in-memory OLAP / BI
- Complex machine learning & graph analysis
- Near real-time, streaming processing
- And many more!

Notices and Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY RELATING TO SALE AND/OR USE OF INTEL PRODUCTS, INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT.

Intel may make changes to specifications, product descriptions, and plans at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

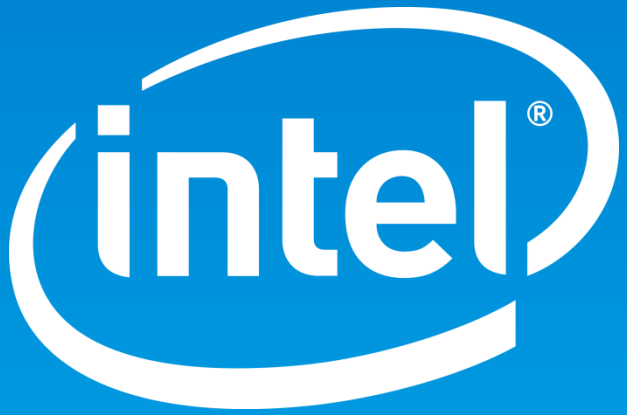
The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

All dates provided are subject to change without notice.

Intel and Intel logoare trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others.


intel