SPARK SUMMIT EUROPE 2017

Indicium: Interactive Querying at Scale

Arvind Heda, Kapil Malik

#EUeco9

What's in the session …

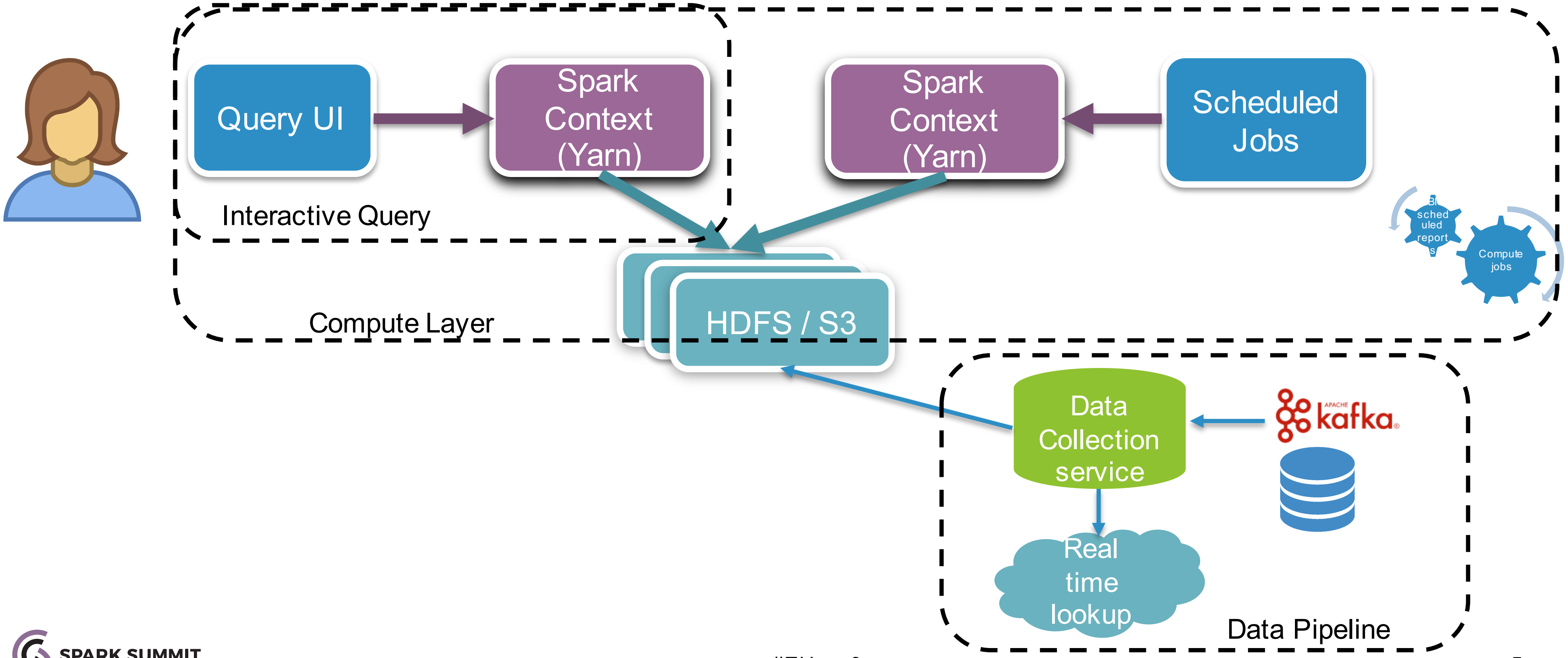
- Unified Data Platform on Spark
  - Single data source for all scheduled / ad hoc jobs and interactive lookup / queries
  - Data Pipeline
  - Compute Layer
  - Interactive Queries?
- Indicium: Part 1 (managed context pool)
- Indicium: Part 2 (smart query scheduler)

Unified Data Platform

Unified Data Platform…(for anything / everything)

- Common Data Lake for storing
  - Transactional data
  - Behavioral data
  - Computed data
- Drives all decisions / recommendations / reporting / analysis from same store.
- Single data source for all Decision Edges, Algorithms, BI tools and Ad Hoc and interactive Query / Analysis tools
- Data Platform needs to support
  - Scale – Store everything from summary to raw data.
  - Concurrency – Handle multiple requests in acceptable user response time.
  - Ad Hoc Drill down to any level – query, join, correlation on any dimension.

Unified Data Platform



Features

Features	Details	Approach
Data Persistence	Store Large Data Volume of Txn, Behavioural and Computed data;	Spark – Parquet format on S3 / HDFS
Data Transformations	Transformation / Aggregation – co relations and enrichments	Batch Processing - Kafka / Java / Spark Jobs
Algorithmic Access	Aggregated / Raw Data Access for scheduled Algorithms	Spark Processes with SQL Context based data access
Decision Making	Aggregated Data Access for decision in real time	In memory cache of aggregated data
Reporting BI / Ad Hoc Query	Aggregated / Raw Data Access for scheduled reports (BI) Aggregated / Raw Data Access for Ad Hoc Queries	BI tool with defined scheduled spark SQL queries on Data store;
Interactive Queries	Drill down data access on BI tools for concurrent users Ad hoc Query / Analysis on data for concurrent users	Scaling challenges for Spark SQL?

Data Pipeline

- Kafka / Sqoop based data collection
- Live lookup store for real time decisions
- Tenant / Event and time based data partition
- Time based compaction to optimize query on sparse data
- Summary Profile data to reduce Joins
- Shared compute resources but different context for Scheduled / Ad Hoc jobs or for Algorithmic / Human touchpoints

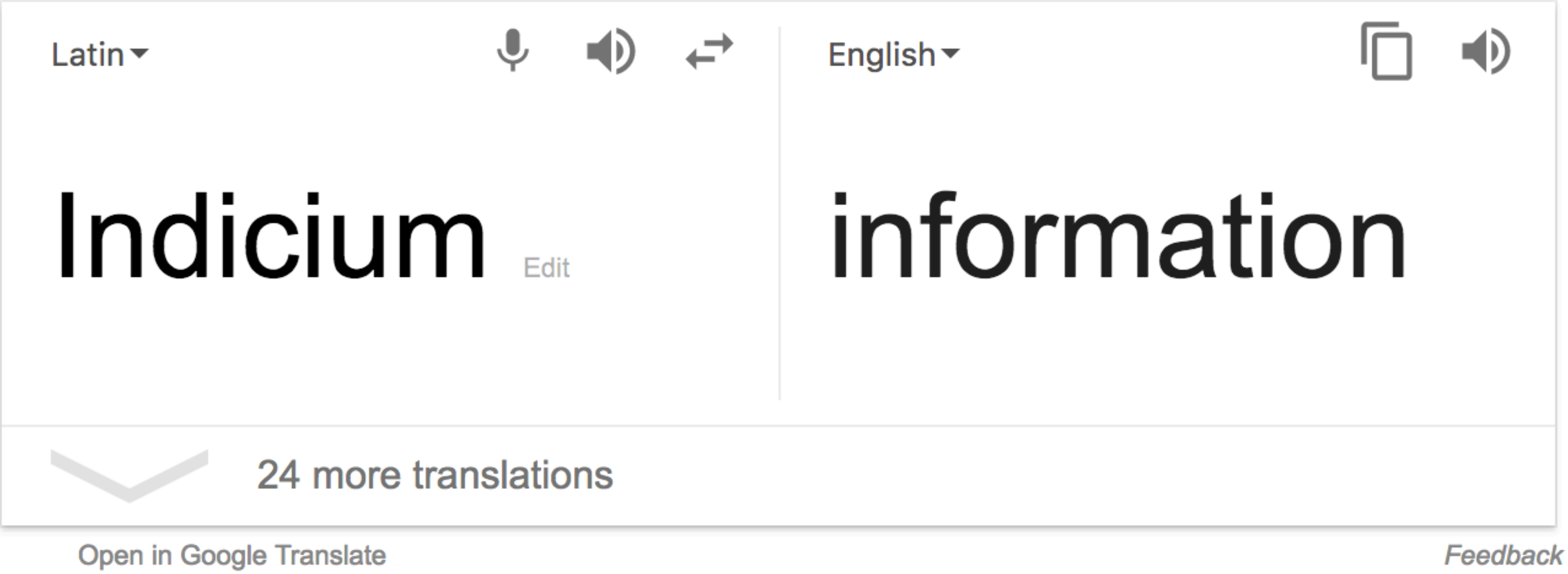
Compute Layer

- No real ‘real time’ queries -- FIFO scheduling for user tasks
- Static or rigid resource allocation between scheduled and ad hoc queries / jobs
- Short lived and stateless context - no sticky ness for user defined views like temp tables.
- Interactive queries ?

What was needed for Interactive query…

- SQL like Query Tool for Ad Hoc Analysis.
- Scalability for concurrent users,
  - Fair Scheduling
  - Responsiveness
- High Availability
- Performance – specifically for scans and Joins
- Extensibility – User Views / Datasets / UDF’s

Indicium ?



Latin

Indicium

English

information

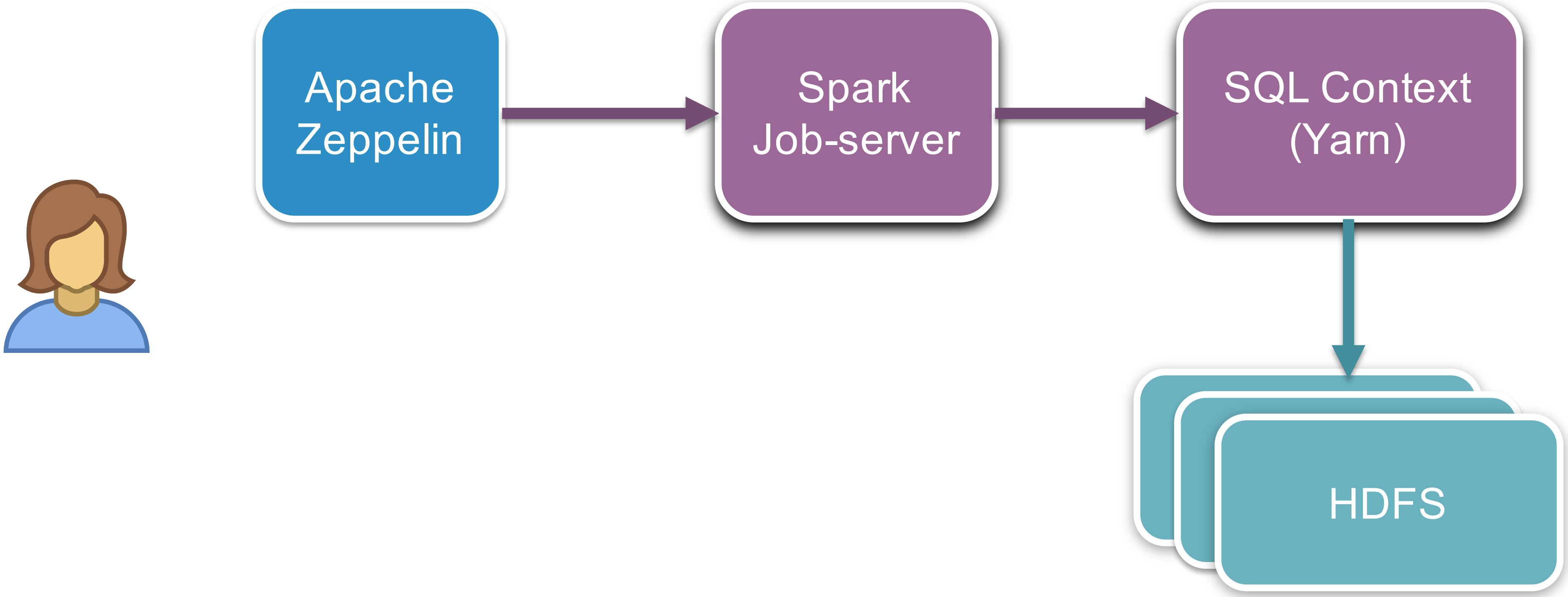
24 more translations

Open in Google Translate

Feedback

Indicium: Part 1
Managed Context Pool

Managed Context Pool



Managed Context Pool

Apache Zeppelin 0.6

- SQL like Query tool and a notebook
- Custom interpreter
  - Configuration: SJS server + context
  - Statement execution: Make asynchronous REST calls to SJS
- Concurrency - Multiple interpreters and notebooks

Spark Job-Server 0.6.x

- Custom SQL context with catalog override
- Custom application to execute queries
- High Availability: Multiple SJS servers and multiple contexts per server

Managed Context Pool

Features

- Familiar SQL interface on notebooks
- Concurrent multi-user support
- Visualization Dashboards
- Long running Spark Job – to support User Defined Views
- Access control on Spark APIs
- Custom SQL context with custom catalog
  - Intercept *lookupTable* calls to query actual data
  - Table wrappers for time windows - like *select count(*) from `lastXDays(table)`*

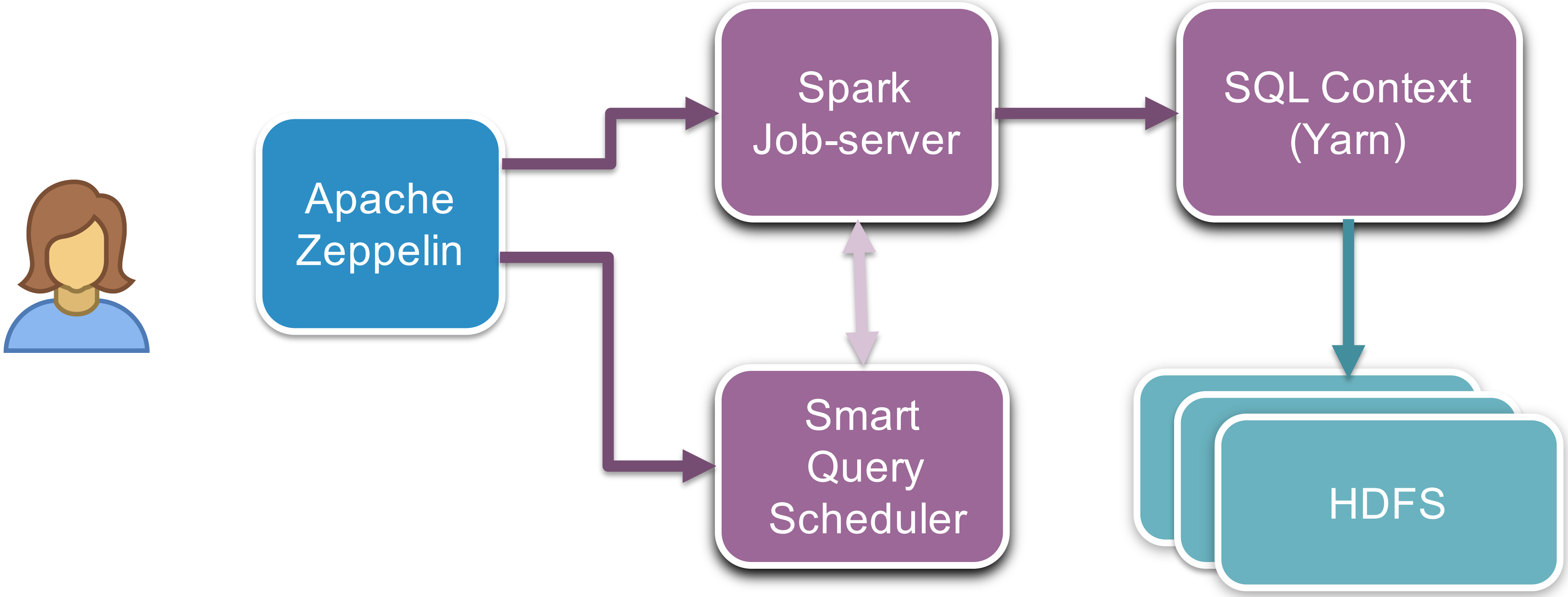
Managed Context Pool

Issues

- Interpreter hard wired to a context
- FIFO scheduling: Single statement per interpreter-context pair – across notebooks / across users
- No automated failure handling
  - Detecting a dead context / SJS server
  - Recovery from the context / server failure
- No dynamic scheduling / load balancing
  - No way of identify an overloaded context
- Incompatible with Spark 2.x

Indicium: Part 2
Smart Query Scheduler

Smart Query Scheduler



Smart Query Scheduler

Zeppelin 0.7

- Supports per notebook statement execution

SJS 0.7 Custom Fork

- Support for Spark 2.x

Smart Query Scheduler:

- Scheduling: API to dynamically bind SJS server + context for every job / query

Other Optimizations:

- Monitoring: Monitor jobs running per context
- Availability: Track Health of SJS servers and contexts and ensures healthy context in pool

Smart Query Scheduler

Dynamic scheduling for every query

- Zeppelin interpreter agnostic of actual SJS / context
- Load balancing of jobs per context
- Query Classification and intelligent routing
- Dynamic scaling / de-scaling the pool size
- Shared Cache
- User Defined Views
- Workspaces or custom time window view for every interpreter

Query Classification / routing

Custom resource configurations for context dedicated for complex or asynchronous queries / jobs:

- Classify queries based on heuristics / historic data into light / heavy queries and route them to different context.
- Separate contexts for interactive vs background queries
  - An export table call does not starve an interactive SQL query

Spark Dynamic Context

Elastic scaling of contexts, co-existing on same cluster as scheduled batch jobs

- Scale up in day time, when user load is high
- Scale down in night, when overnight batch jobs are running
- Scaling also helped to create reserved bandwidth for any set of users, if needed.

Shared Cache

Alluxio to store common datasets

- Single cache for common datasets across contexts
  - Avoids replication across contexts
  - Cached data safe from executor / context crashes
- Dedicated refresh thread to release / update data consistently across contexts

Persistent User Defined Views

- Users can define a temp view for a SQL query
- Replicated across all SJS servers + contexts
- Definitions persisted in DB so that a context restart is accompanied by temp views' registration.
- Load on start to warm up load of views
- TTL support for expiry

Workspaces

- Support for multiple custom catalogs in SQL context for table resolution
- Custom time range / source / caching
  - Global
  - Per catalog
  - Per table
- Configurable via Zeppelin interpreter
- Decoupled time range from query syntax
  - Join a behavior table(refer to last 30 days) with lookup table (fetch complete data)

Automated Pool Management

- Monitoring scripts to track and restart unhealthy / un-responsive SJS servers / contexts
- APIs on SJS to stop / start / refresh context / SJS
- APIs to refresh cached tables / views;
- APIs on Router Service to reconfigure routing / pool size and resource allocation

Thank You !

Questions & Answers

kapil.ee06@gmail.com
arvind_heda@yahoo.com

References

- Apache Zeppelin: https://zeppelin.apache.org/
- Spark Job-server: https://github.com/spark-jobserver/spark-jobserver
- Alluxio: http://www.alluxio.org/

Scale ….

- Data
  - ~ 100 TB
  - ~ 1000 Event Types
- 100+ Active concurrent users
- 30+ Automated Agents
- 10000+ Scheduled / 3000+ Ad Hoc Analysis
- Avg data churn per Analysis > 200 GB