TSVD 11

Technológie spracovania veľkých dát

Peter Bednár, Martin Sarnovský

Obsah

- Big data tools
  - Batch processing
  - Streaming
  - Coordination and management

Batch processing

Apache Hive

- Nástroj typu data warehouse nad Hadoop ekosystémom, umožňuje analyzovať veľké objemy dát pomocou SQL-like jazyka (*HiveQL*)
  - zjednodušuje prácu oproti priamemu použitiu MapReduce
- Motivácia - umožniť analytikom pracovať s dátami bez potreby programovania distribuovaných algoritmov, pričom systém automaticky prekladá dotazy na vykonateľné joby
- Optimalizovaný pre OLAP scenáre (analytické dopyty nad veľkými datasetmi), kde sa často vykonávajú agregácie a množinové operácie
- **Nie je** určený pre OLTP (transakčné spracovanie), čo znamená, že nie je vhodný pre časté aktualizácie alebo nízku latenciu dotazov
- Typické použitie Hive zahŕňa reporting, business intelligence a dávkové analytické úlohy nad dátami uloženými v HDFS.

Hive - architektúra

- **Metastore** - slúži na ukladanie metadát o tabuľkách, partíciách a dátových typoch, pričom tieto informácie sú uložené v relačnej databáze a používajú sa pri plánovaní dotazov.
- **HiveQL engine** - zabezpečuje parsovanie a validáciu SQL dotazov, pričom ich transformuje do logického plánu, ktorý reprezentuje operácie ako join, filter alebo agregácia.
- **Execution engine** - následne prekladá logický plán na fyzický plán (napr. MapReduce DAG), ktorý je distribuovane vykonaný na klastri.
- **Storage layer** - využíva HDFS alebo HBase na ukladanie samotných dát, pričom Hive pracuje nad týmito dátami bez ich presunu do vlastného úložiska.
- Architektúra - navrhnutá tak, aby oddelila logickú vrstvu (dopyty) od fyzickej implementácie
  - umožňuje flexibilitu pri zmene backendu (napr. z MapReduce na Spark).

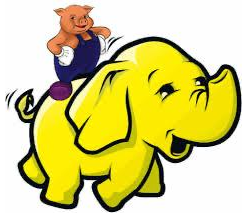
Hive – dátový model

- Organizuje dáta do tabuliek, ktoré majú podobnú štruktúru ako tabuľky v relačných databázach
  - každá tabuľka je reprezentovaná adresárom v HDFS
- **Partície** umožňujú rozdeliť tabuľku na menšie logické časti podľa hodnoty atribútu (napr. dátum), čím sa výrazne znižuje objem dát, ktoré je potrebné spracovať pri dopyte
- **Buckets** - predstavujú ďalšie rozdelenie dát na základe hash funkcie, čo zlepšuje paralelizáciu a výkon operácií ako *join* alebo *sampling*
- Hive podporuje nielen základné dátové typy (*int*, *string*), ale aj komplexné štruktúry ako mapy, polia a štruktúry, čo umožňuje pracovať so semi-štruktúrovanými dátami.
- Tento dátový model je navrhnutý pre efektívne sekvenčné čítanie veľkých datasetov, nie pre rýchly prístup k jednotlivým záznamom.

Hive - zhodnotenie

- Výhody
  - jednoduché SQL rozhranie, ktoré umožňuje analytikom pracovať s big data bez znalosti distribuovaného programovania, čo výrazne znižuje bariéru vstupu.
  - vysoká škálovateľnosť (využíva Hadoop infraštruktúru), zvláda spracovanie petabajtov dát na veľkých klastroch
- Nevýhody
  - vysoká latencia, keďže dotazy sú vykonávané ako batch joby, čo znamená, že nie sú vhodné pre interaktívne alebo real-time scenáre
  - nepodporuje plnohodnotné transakcie ani komplexné operácie ako join v rovnakom rozsahu ako relačné databázy, čo obmedzuje jeho použitie v niektorých prípadoch
- Vhodný najmä pre offline analýzy, reporting a ETL procesy, kde nevadí vyššia latencia spracovania

Apache Pig



- Nástroj pre spracovanie veľkých dát, ktorý poskytuje skriptovací jazyk *Pig Latin* na definovanie dátových transformácií
  - zjednodušuje tvorbu MapReduce úloh
- Navrhnutý ako reakcia na zložitosť MapReduce
  - aj jednoduché operácie vyžadovali veľké množstvo kódu, čo spomaľovalo vývoj a zvyšovalo chybovosť
- Na rozdiel od Hive, ktorý je deklaratívny (SQL) - Pig je **procedurálny** jazyk, kde programátor explicitne definuje tok dát a jednotlivé transformácie
- Vhodný najmä pre data engineering úlohy, kde je potrebné kombinovať viacero krokov spracovania do jednej pipeline
- Výsledný Pig skript je preložený na sekvenciu MapReduce jobov, ktoré sú vykonané v Hadoop klastri

Pig – základný princíp

- *Pig skript* predstavuje sekvenciu operácií nad dátami, kde každá operácia transformuje vstupné dáta na výstupné, čím vzniká dátový tok (dataflow)
- Operácie (*filter*, *group*, *join* atď.) sú automaticky mapované na MapReduce joby, čo znamená, že programátor nemusí riešiť paralelizáciu
- *Pig engine* analyzuje skript, optimalizuje ho a vytvára logický plán, ktorý je následne preložený do fyzického plánu vykonania
- Výhoda - možnosť kombinovať viacero transformácií v jednom skripte, čo znižuje počet samostatných jobov a zlepšuje výkon
- Podpora používateľom definované funkcií, čo umožňuje rozšíriť jeho funkcionalitu o vlastné operácie

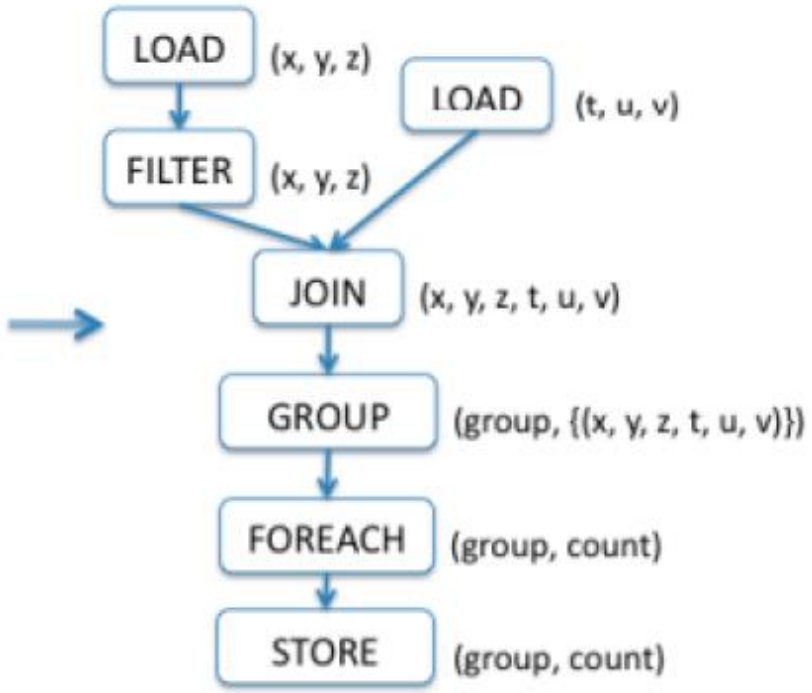
Pig - dátový model

- Flexibilný dátový model
  - Základné typy *atom*, *tuple*, *bag, map* atď., umožňuje pracovať aj s neštruktúrovanými dátami
- **Tuple** reprezentuje záznam (riadok), podobne ako v relačnej databáze, ale môže obsahovať rôzne typy dát
- **Bag** predstavuje kolekciu tuple-ov, čo je ekvivalent tabuľky, ale bez pevnej schémy
- **Map** umožňuje ukladať páry kľúč/hodnota, čo je vhodné pre semi-štruktúrované dáta
- Tento flexibilný model robí Pig vhodným pre ETL úlohy, kde sa často pracuje s rôznorodými dátovými formátmi

Pig - ukážka

Pig Latin

```
A = LOAD 'file1' AS (x, y, z);
B = LOAD 'file2' AS (t, u, v);
C = FILTER A by y > 0;
D = JOIN C BY x, B BY u;
E = GROUP D BY z;
F = FOREACH E GENERATE
    group, COUNT(D);
STORE F INTO 'output';
```



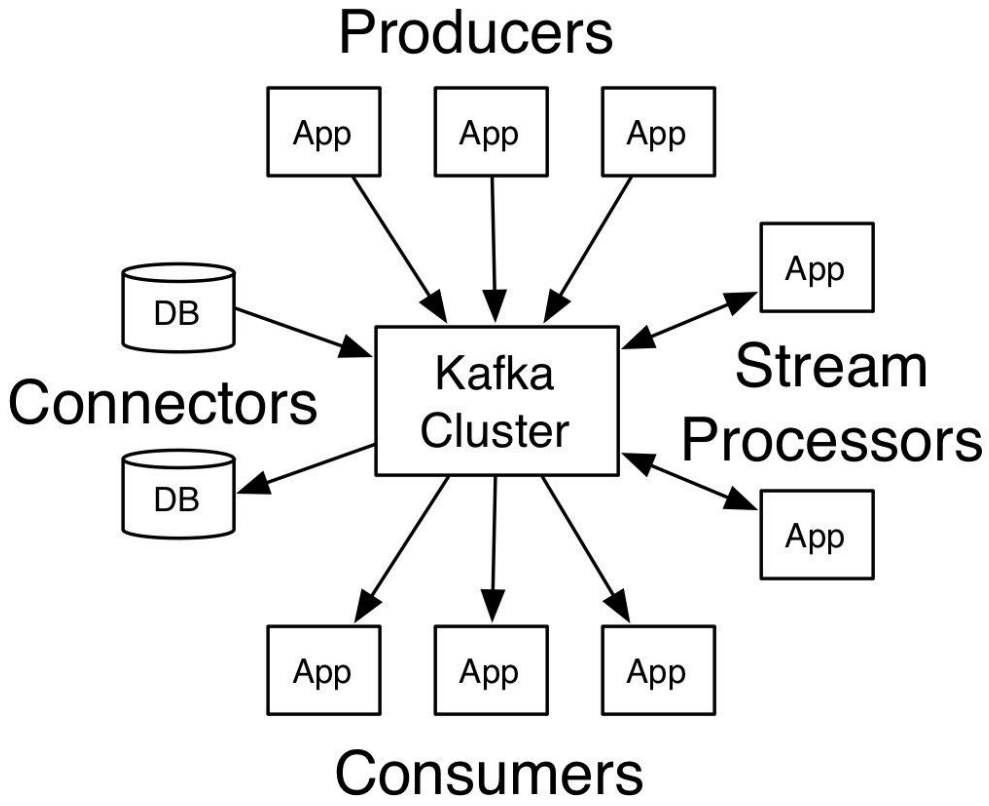
Pig - zhodnotenie

- Výhody
  - jednoduchšie programovanie oproti MapReduce, čo umožňuje rýchlejší vývoj dátových pipeline
  - vhodný pre komplexné transformácie dát, kde je potrebné explicitne definovať tok dát a operácie nad nimi
- Nevýhody
  - stále ide o batch spracovanie, takže nie je vhodný pre real-time scenáre
  - pre analytikov môže byť menej intuitívny ako Hive, keďže nepoužíva SQL syntax, ale procedurálny jazyk
- Používa sa dnes menej ako moderné nástroje, ale konceptuálne je dôležitý pre pochopenie dataflow prístupov

Stream processing

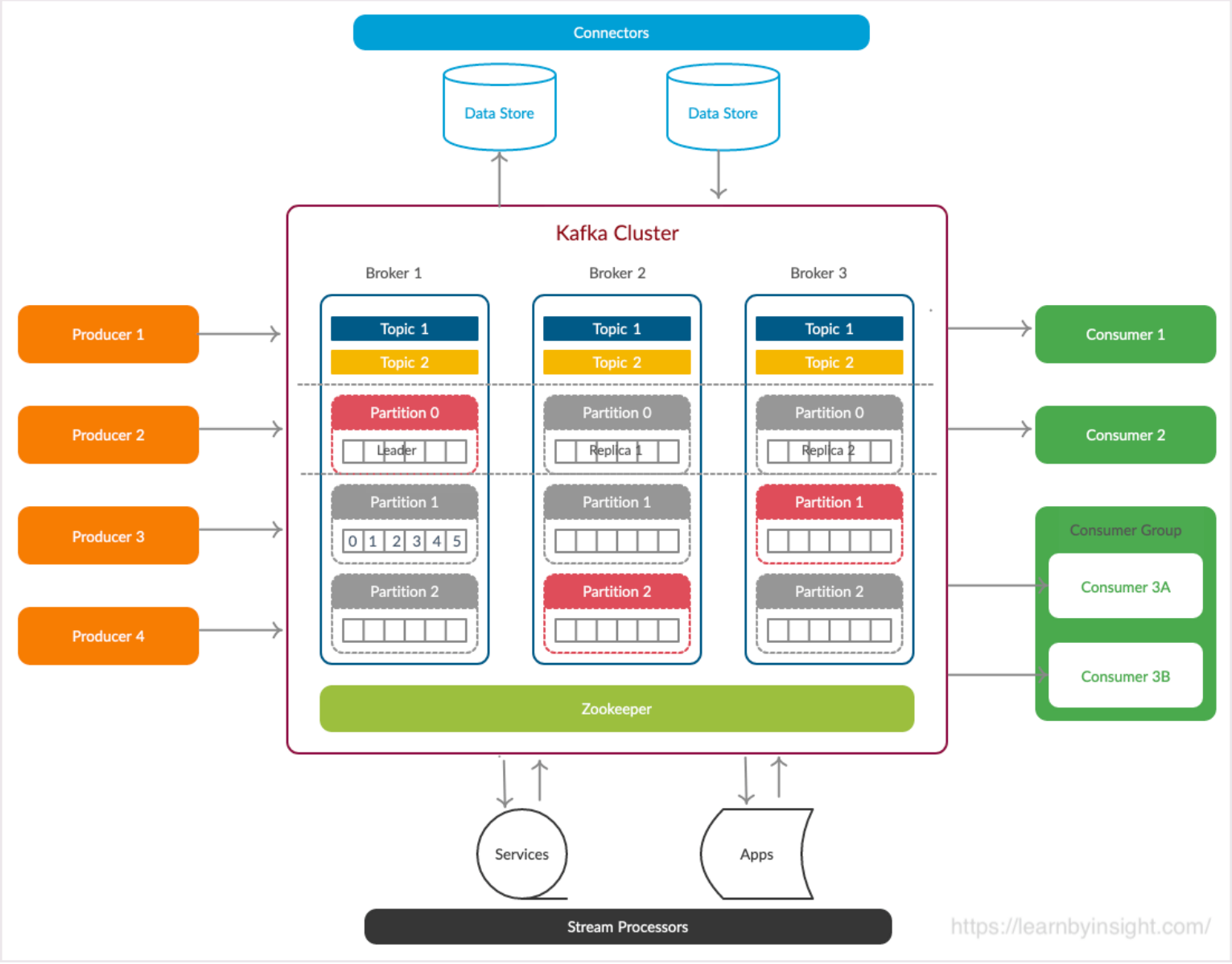
Apache Kafka

- Distribuovaný messaging systém založený na publish-subscribe modeli, umožňuje prenos veľkého množstva správ medzi systémami v reálnom čase
- *Producenti* zapisujú správy do Kafka tém (*topics)*, zatiaľ čo *konzumenti* tieto správy čítajú podľa vlastného tempa, čo umožňuje asynchrónne spracovanie
- *Log-based systém, správy* uchovávané po určitý čas, nezávisle od toho, či boli spracované konzumentmi
- Systém navrhnutý pre vysokú priepustnosť a nízku latenciu– vhodný pre streaming scenáre ako logovanie, monitoring alebo event processing
- V praxi často slúži ako „centrálny buffer“ medzi systémami, napríklad medzi zberom dát a ich spracovaním (napr. Spark Streaming)



Kafka - architektúra

- Dáta sú organizované do **topics**, ktoré sú ďalej rozdelené na **partitions**, čo umožňuje paralelné spracovanie a škálovanie systému
- Kafka cluster pozostáva z **brokerov**, pričom každý broker spravuje časť dát a môže byť leaderom pre niektoré partitions
- Každá partition je replikovaná na viacerých uzloch, čo zabezpečuje odolnosť voči zlyhaniam a vysokú dostupnosť
- Konzumenti sledujú svoju pozíciu v logu (offset), čo im umožňuje opätovné spracovanie dát alebo paralelné čítanie
- Tento model umožňuje horizontálne škálovanie pridaním ďalších brokerov a partitions



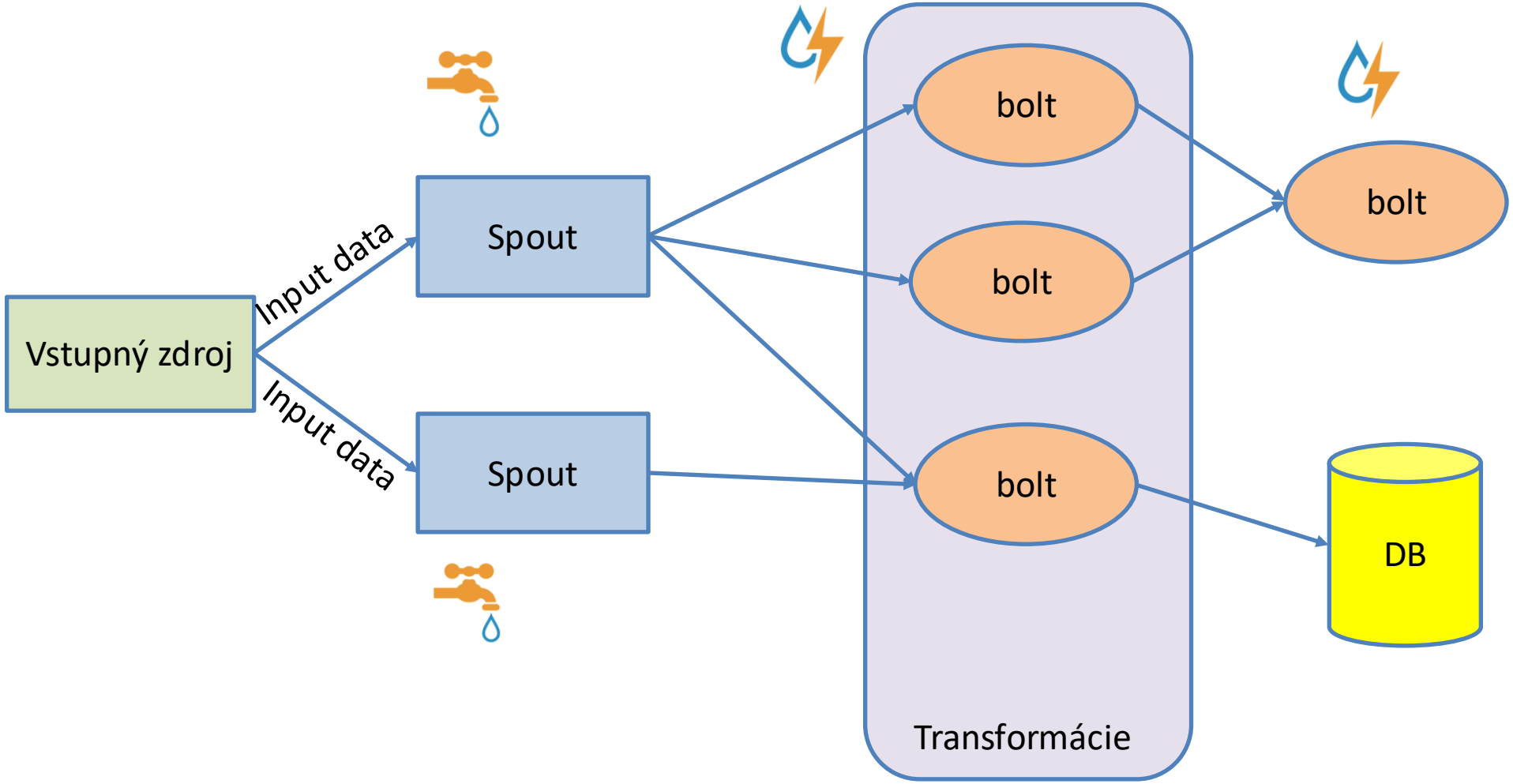
Apache Storm

- Distribuovaný systém pre spracovanie dátových prúdov v reálnom čase, umožňuje kontinuálne spracovanie dát bez potreby ich ukladania do dávok.
- Na rozdiel od batch systémov spracováva každú správu okamžite po jej príchode, čo umožňuje dosiahnuť veľmi nízku latenciu (rádovo milisekundy až sekundy).
- Navrhnutý ako fault-tolerant systém, ktorý garantuje doručenie správ (napr. „at least once“), čo je dôležité pre kritické aplikácie.
- Používa sa najmä pre real-time analytiku, monitoring, detekciu anomálií alebo spracovanie eventov v reálnom čase.
- V súčasnosti je Storm často nahrádzaný novšími systémami (napr. Flink), ale jeho model je dôležitý pre pochopenie stream processingu.

Storm – model spracovania

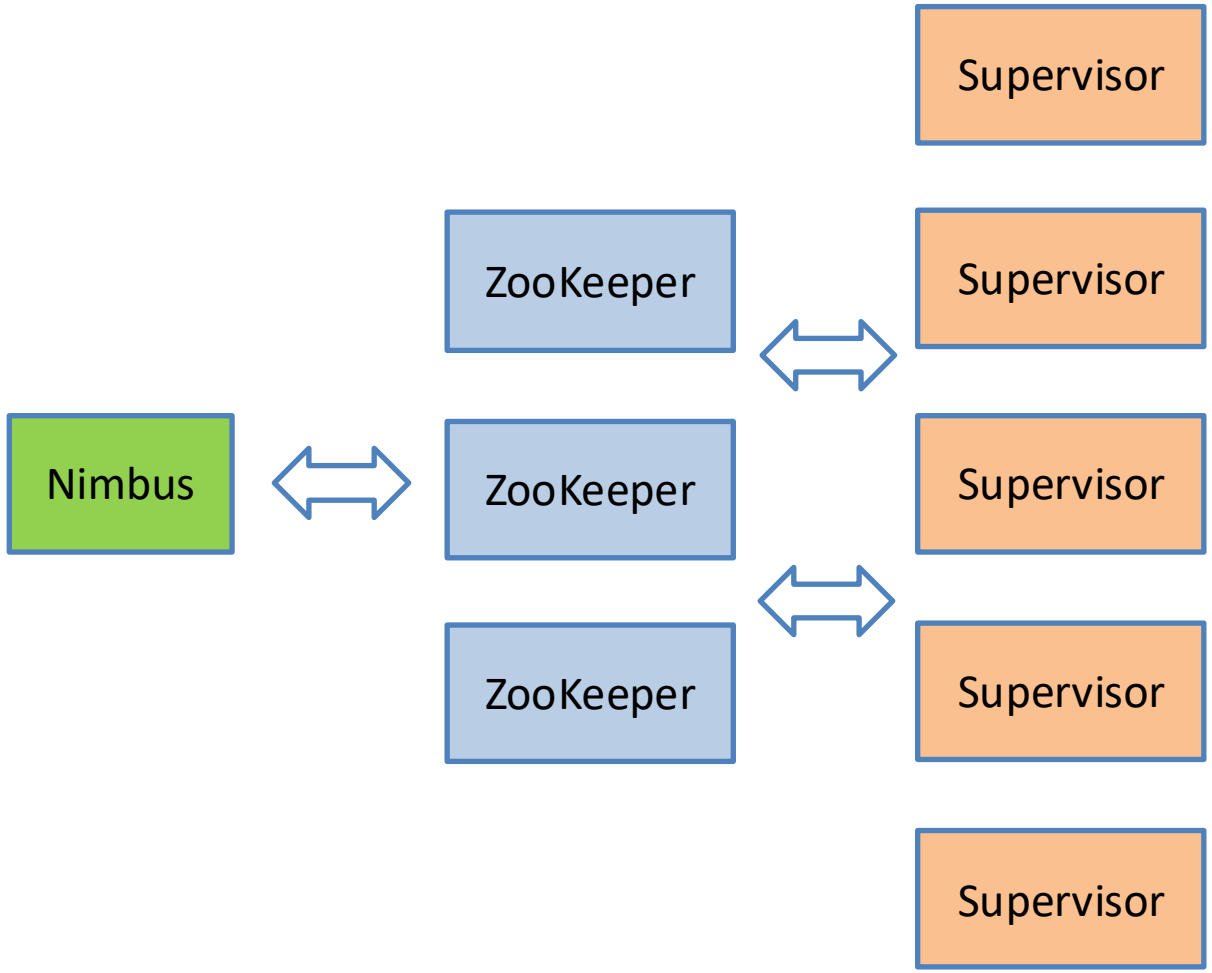
- Storm spracovanie je definované ako **topológia**, čo je orientovaný graf, kde uzly reprezentujú výpočty a hrany dátové toky medzi nimi.
- **Spout** - predstavuje zdroj dát (napr. Kafka, API), ktorý generuje prúd správ a posiela ich do systému.
- **Bolt** - vykonáva transformácie nad dátami, ako napríklad filtrovanie, agregácia, join alebo ukladanie výsledkov.
- Dáta sú reprezentované ako **tuples**, ktoré sa prenášajú medzi jednotlivými komponentmi topológie.
- Tento model umožňuje vytvárať flexibilné a škálovateľné pipeline pre spracovanie dát v reálnom čase.

Storm – model spracovania



Storm – architektúra

- **Nimbus** je hlavný uzol (master), ktorý riadi vykonávanie topológií, distribuuje úlohy a monitoruje ich stav v klastri.
- **Supervisor** beží na pracovných uzloch a je zodpovedný za spúšťanie a riadenie worker procesov, ktoré vykonávajú samotné výpočty.
- **ZooKeeper** zabezpečuje koordináciu medzi uzlami, uchováva stav systému a pomáha pri zotavení zo zlyhaní.
- Storm umožňuje horizontálne škálovanie pridaním ďalších worker uzlov, čím sa zvyšuje paralelizmus spracovania.
- Architektúra je navrhnutá tak, aby systém vedel pokračovať v spracovaní aj pri zlyhaní jednotlivých uzlov.

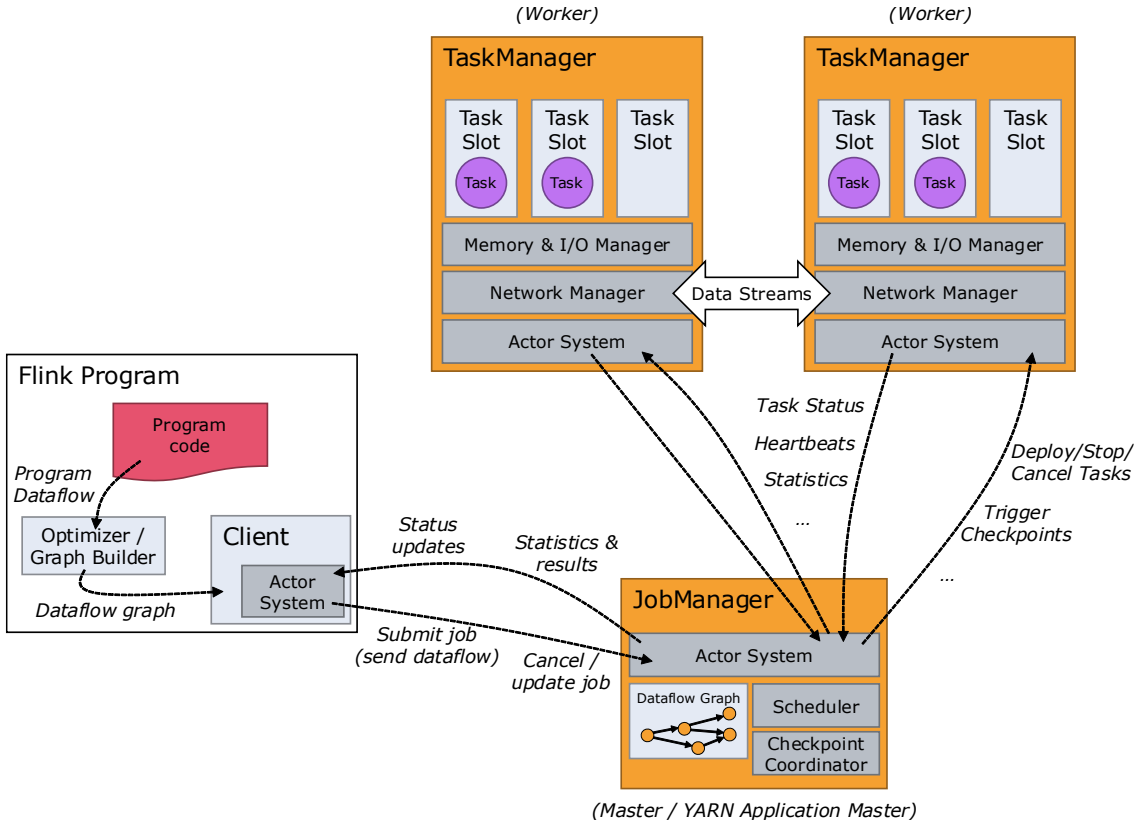


Apache Flink

- Moderný distribuovaný systém pre spracovanie dátových prúdov, ktorý podporuje *skutočný stream processing* (**nie micro-batch** ako Spark).
- Spracováva dáta ako nekonečné prúdy udalostí, pričom umožňuje pracovať aj s historickými dátami ako špeciálnym prípadom streamu.
- Podpora **event-time spracovania**, čo znamená, že operácie sú založené na čase vzniku udalosti, nie na čase jej spracovania.
- Poskytuje presné garancie spracovania („exactly-once semantics“), čo je kritické pre finančné alebo analytické aplikácie.
- V súčasnosti patrí medzi najpoužívanejšie nástroje pre real-time analytiku vo veľkých systémoch.

Flink - architektúra

- **JobManager** – zodpovedný za plánovanie úloh, rozdelenie jobu na menšie paralelné jednotky (tasks) a ich distribúciu na jednotlivé TaskManagery, pričom zároveň monitoruje ich stav a rieši zlyhania.
- **TaskManager** – beží na pracovných uzloch a vykonáva konkrétne operácie nad dátami (napr. map, filter, window), má definovaný počet slotov, ktoré určujú mieru paralelizmu.
- Spracovanie prebieha ako **DAG (Directed Acyclic Graph)** operátorov, každý uzol reprezentuje operáciu nad streamom a hrany reprezentujú dátové toky medzi operátormi
  - Job Graph (logická úroveň) a Execution Graph (tasky)
- DAG umožní
  - paralelizáciu (viac inštancií jedného operátora)
  - optimalizáciu (napr. chaining operátorov)
- **Pipeline execution -** dáta pretekajú medzi operátormi kontinuálne bez potreby materializácie na disk - znižuje latenciu oproti batch
- **Stateful processing** - operátory uchovávajú stav (napr. agregácie, počítadlá) uložený lokálne a pravidelne checkpointovaný.
- **Checkpointing** - periodicky ukladá konzistentný snapshot stavu aplikácie, čo umožňuje obnoviť výpočet po zlyhaní bez straty dát (exactly-once semantics).
- Podporuje **event-time spracovanie a watermarky**, ktoré umožňujú korektne spracovať aj oneskorené udalosti v dátových prúdoch.



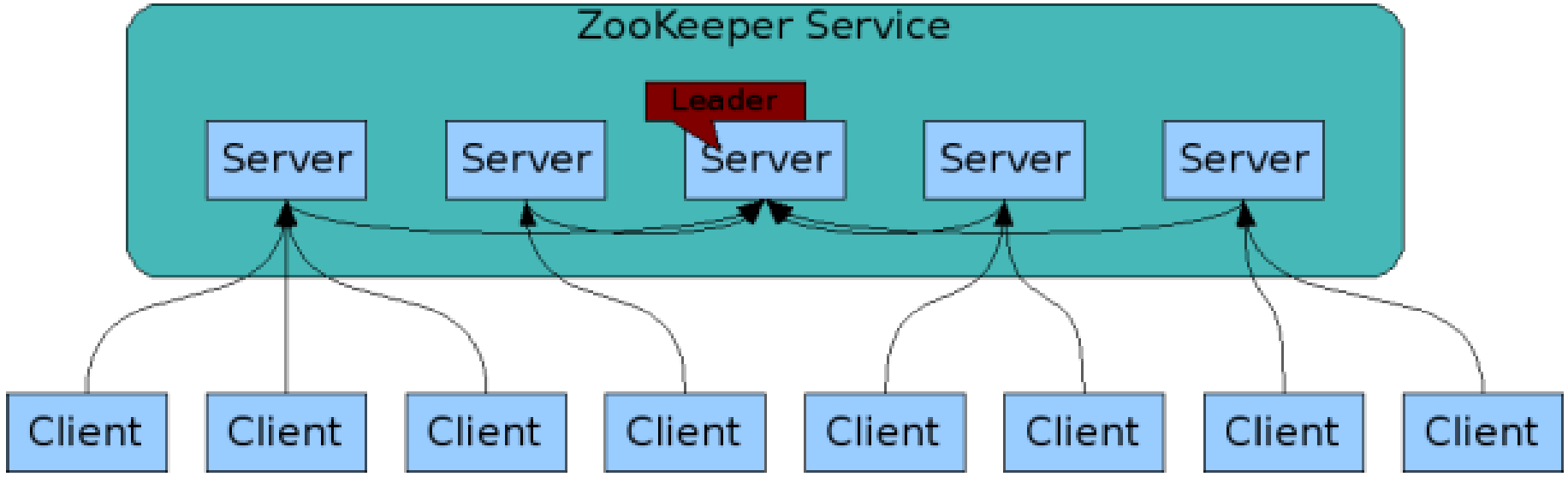
Coordination and management

Apache Zookeeper

- Distribuovaná služba, ktorá zabezpečuje koordináciu medzi uzlami v klastri, napríklad pri leader election, konfigurácii alebo synchronizácii procesov.
- Dáta sú uložené v hierarchickej štruktúre nazývanej **znodes**, ktoré fungujú podobne ako súborový systém a umožňujú zdieľať stav medzi rôznymi komponentmi systému.
- ZooKeeper poskytuje mechanizmus **watchers**, ktorý umožňuje klientom sledovať zmeny v stave systému a reagovať na ne (napr. zmena leadera).
- Systém funguje na princípe **leader-follower replikácie**, kde leader spracováva zápisy a zabezpečuje konzistenciu medzi uzlami.
- ZooKeeper sa používa v mnohých big data technológiách (Kafka, Storm, HBase) ako základná vrstva pre koordináciu a správu clusteru.

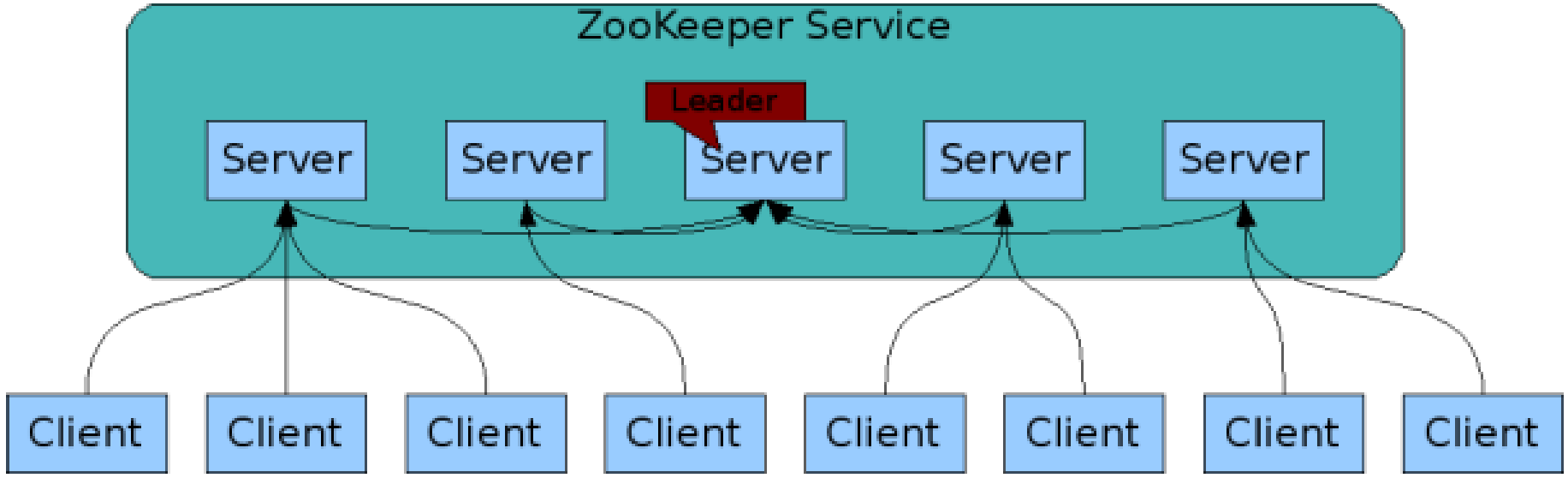
ZooKeeper

- ZooKeeper – distribuovaná služba, replikovaná na viacerých uzloch klastra
- Jeden zo ZooKeeper uzlov je zvolený do role Leader-a
- Hlavnou úlohou Leader-a – zoradiť požiadavky klientov, ktorí menia stav (napr. zapisujú dáta od klientov)



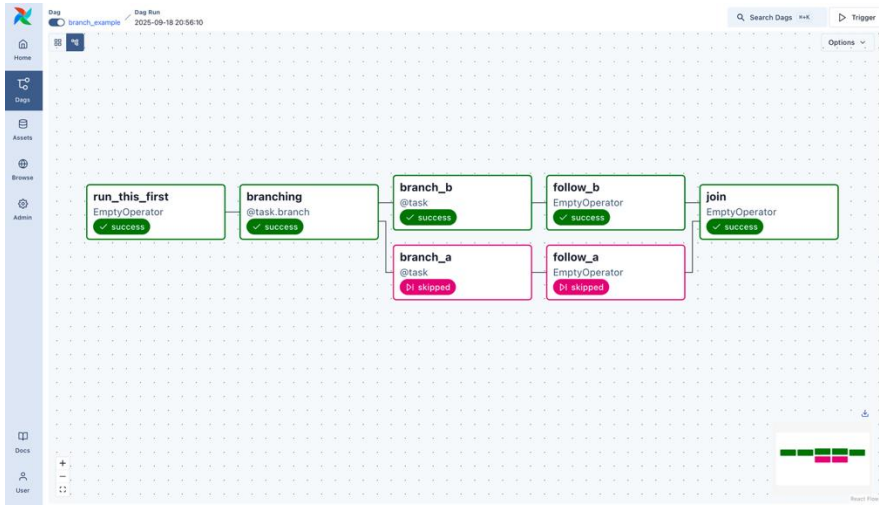
ZooKeeper

- Každý ZooKeeper server uchováva znodes hierarchiu v pamäti (kvôli nízkej latencii)
- Klient sa pripája na jeden zo ZooKeeper serverov a udržiava s ním konektivitu
- ZooKeeper služba je dostupná, keď je dostupná väčšina ZooKeeper serverov



Apache Airflow

- Nástroj pre orchestráciu a plánovanie dátových pipeline, ktorý umožňuje definovať workflow ako *DAG úloh*, ktoré sa vykonávajú podľa závislostí.
- Pipeline v Airflow sú definované v Pythone - flexibilné programovanie workflow a jednoduchú integráciu s rôznymi systémami (napr. Hive, Spark, Kafka, Flink).
- Každá úloha (task) v DAG reprezentuje konkrétny krok pipeline (napr. načítavanie dát, transformácia, trénovanie modelu), Airflow riadi ich poradie a spúšťanie.
- Airflow obsahuje *scheduler*, ktorý spúšťa úlohy podľa časového plánu alebo udalostí, a zároveň monitoruje ich úspešnosť a retry mechanizmy.
- Poskytuje webové rozhranie, kde je možné vizualizovať pipeline, sledovať ich stav a debugovať problémy.
- Airflow je často používaný ako „lepidlo“ medzi jednotlivými nástrojmi v big data ekosystéme, napríklad na orchestráciu batch a streaming pipeline.



Apache Ambari