_РАЗНОЕ

4 февраля 2021 г. 12:28

[* эволюция архитектур]

[Archiving events]

пакетная и потоковая передача не так уж сильно отличаются:

- это всего лишь незначительные вариации потоков и таблиц тема. Как мы узнали в главе 6 , основное различие действительно сводится к возможности инкрементно запускать таблицы в потоки; все остальное концептуально то же самое. 1
- Воспользовавшись общностью, лежащей в основе этих двух подходов, можно было обеспечить единый, почти непрерывный опыт, применимый к обоим мирам. Это был большой шаг вперед в повышении доступности потоковой обработки.

ключевые аспекты

- *Unaligned, event-time windows* such as sessions, providing the ability to concisely express powerful analytic constructs and apply them to out-of-order data.
- *Custom windowing support*, because one (or even three or four) sizes rarely fit all.
- *Flexible triggering* and *accumulation modes*, providing the ability to shape the way data flow through the pipeline to match the correctness, latency, and cost needs of the given use case.
- The use of *watermarks* for reasoning about *input completeness*, which is critical for use cases like anomalous dip detection where the analysis depends upon an absence of data.
- *Logical abstraction* of the underlying execution environment, be it batch, microbatch, or streaming, providing flexibility of choice in execution engine and avoiding system-level constructs (such as micro-batch size) from creeping into the logical API.

Многие распределенные системы обработки потоков с открытым исходным кодом разработаны с расчетом на аналитику:

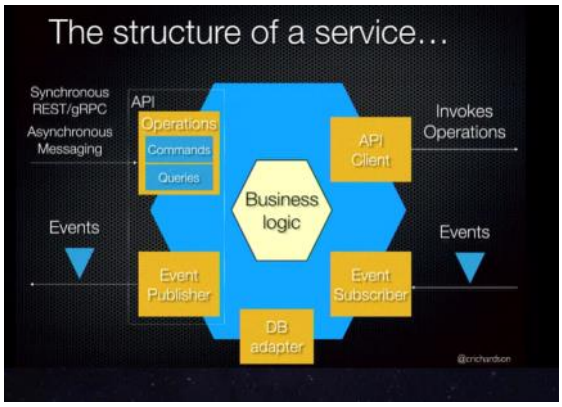
- например,
- Apache Storm,
- Spark Streaming,
- Flink,
- Concord,
- Samza,
- Kafka Streams [74];

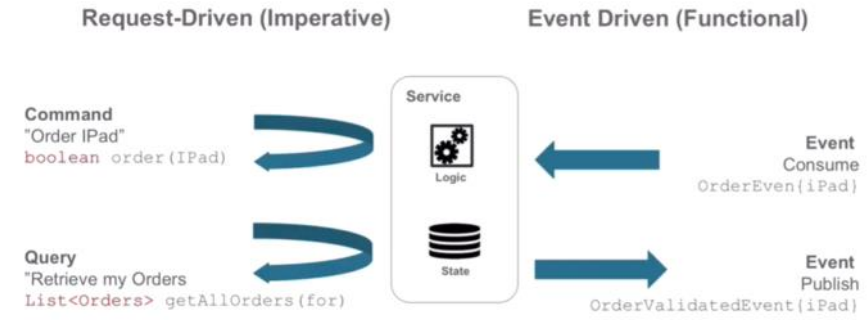
- есть также хостинговые сервисы, такие как
- Google Cloud Dataflow и
- Azure Stream Analytics.

ЭВОЛЮЦИЯ АРХИТЕКТУР

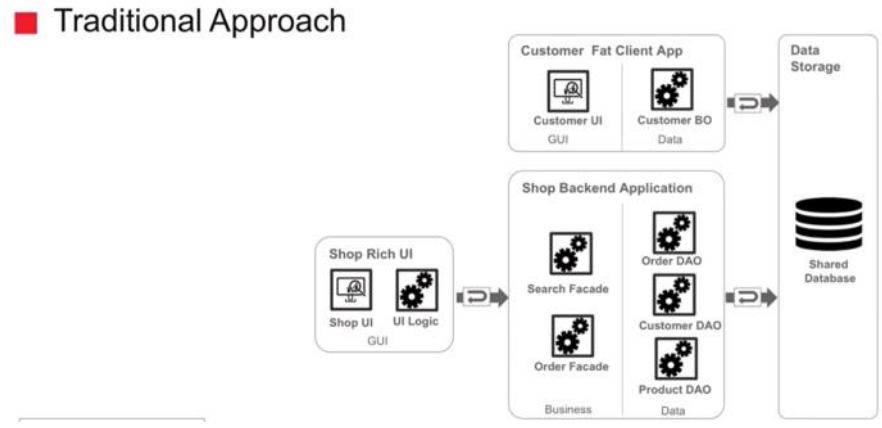
14 октября 2020 г. 14:30

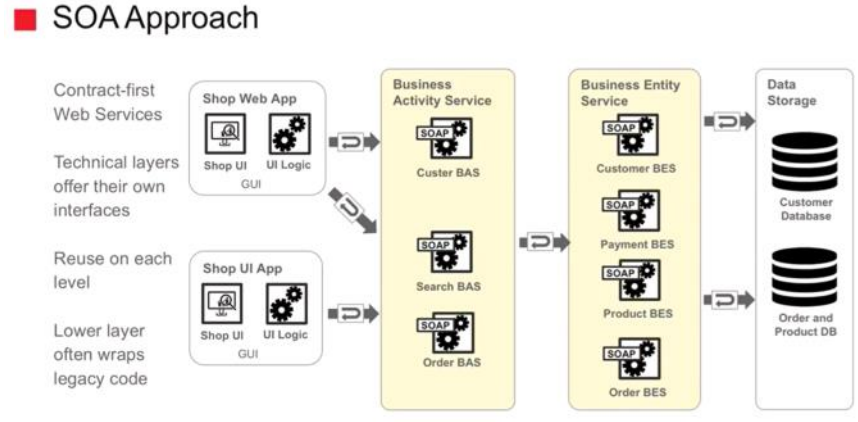
чем отличаюся команды/запросы и события



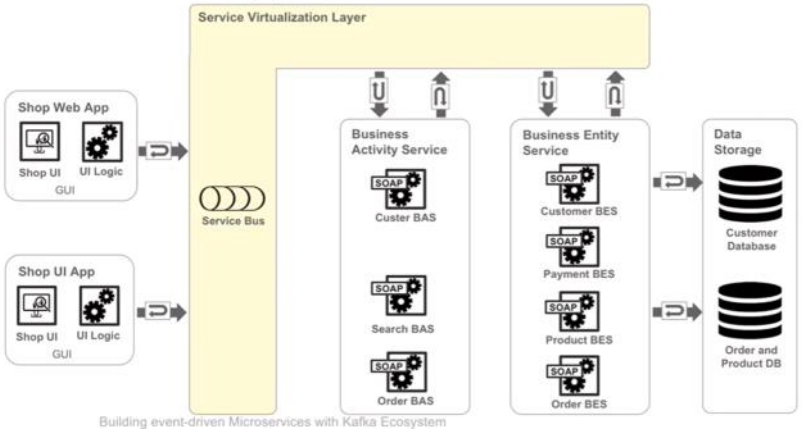


эволюция архитектур





Virtualized SOA Approach



Microservice Approach

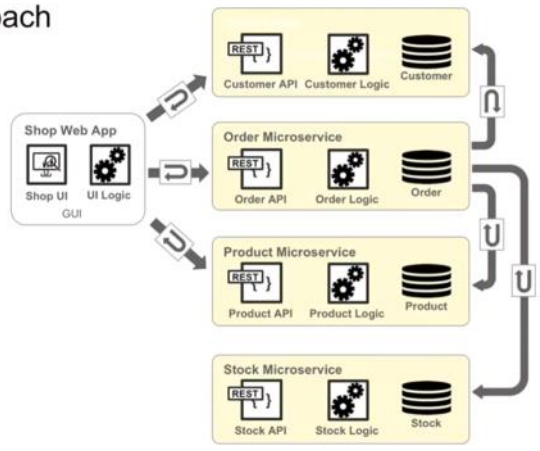
Tightly Scoped behind clear interfaces

Responsible for managing their own data (not necessarily the infrastructure)

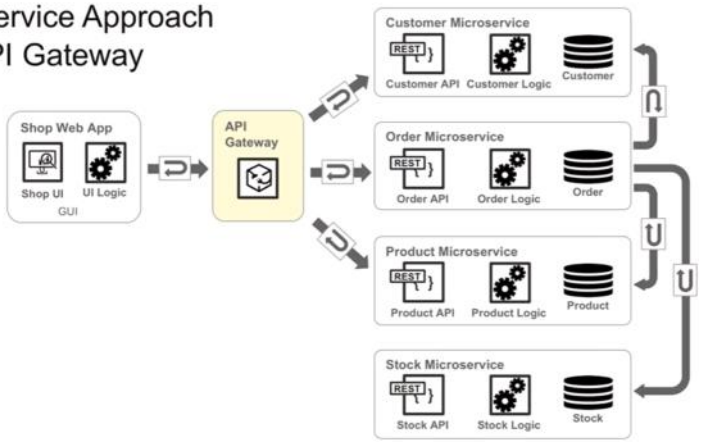
highly decoupled

Independently deployable, self-contained and autonomous

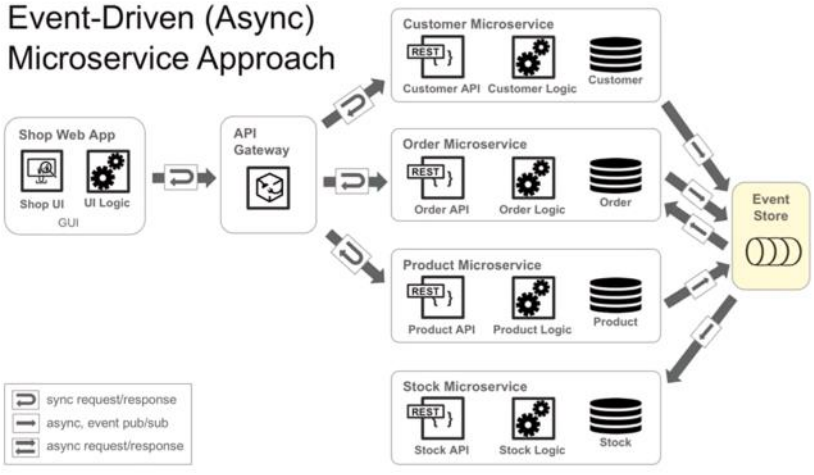
SOA done right ?!



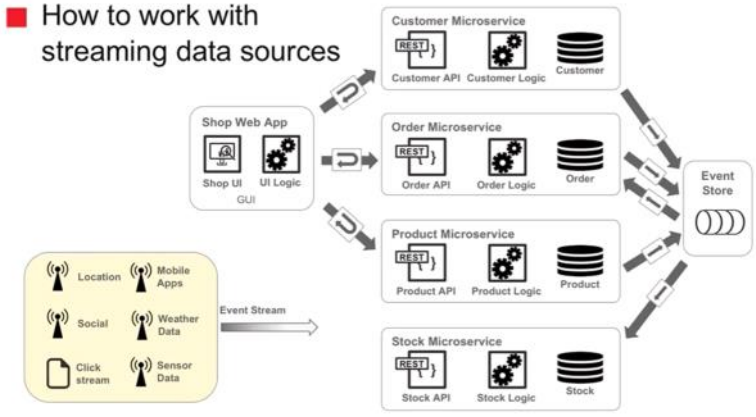
Microservice Approach with API Gateway

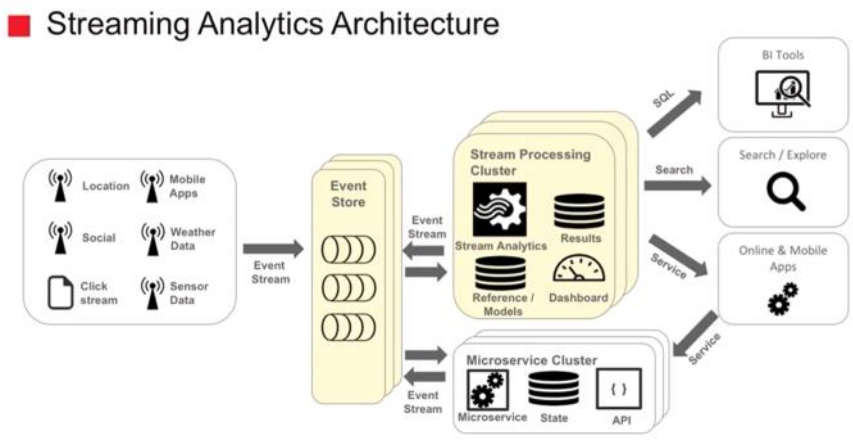


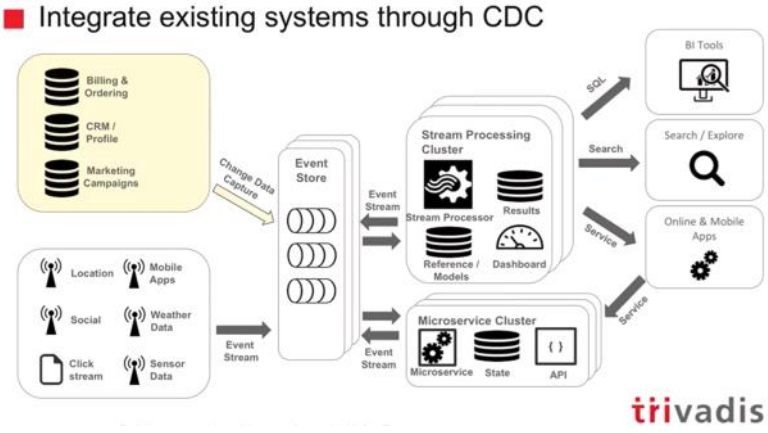
Event-Driven (Async) Microservice Approach

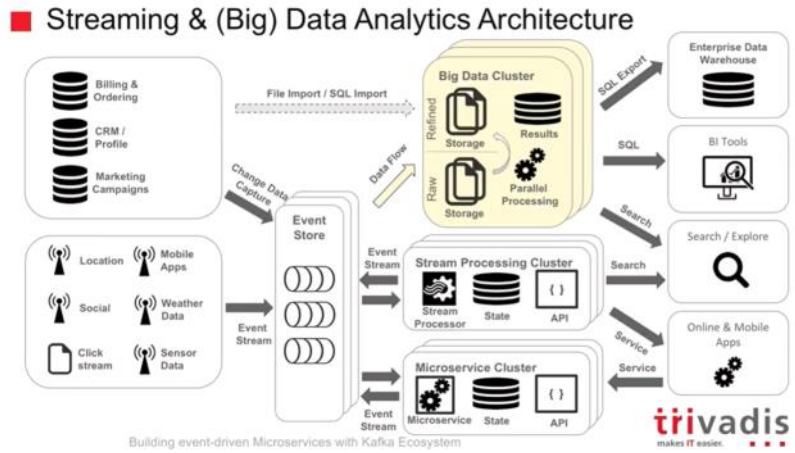


разное Streaming



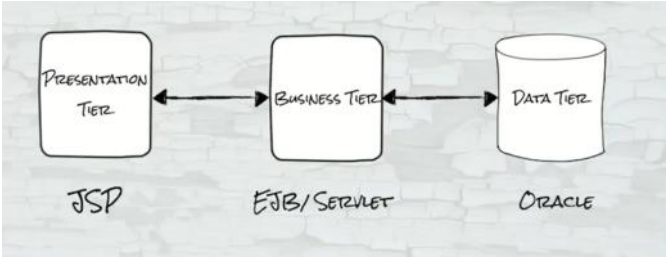




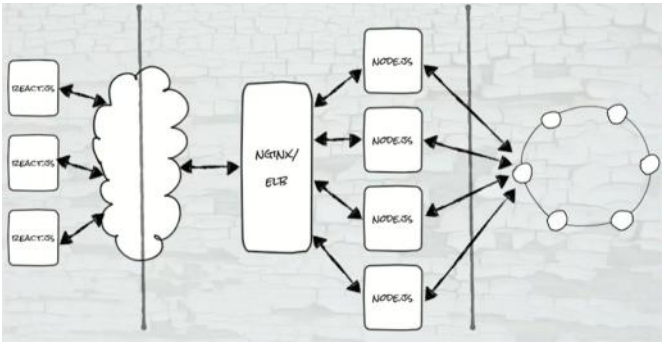


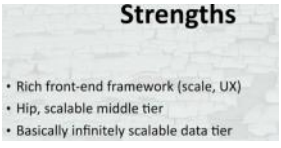
три уровня до сих пор остались

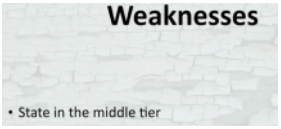
(1) обычная трехуровневая архитектура (раньше)



(2) обычная трехуровневая архитектура (сейчас)

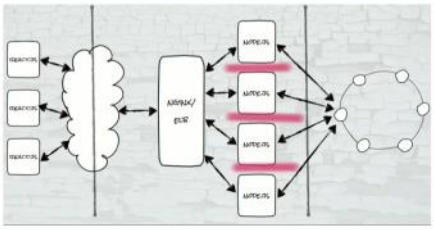




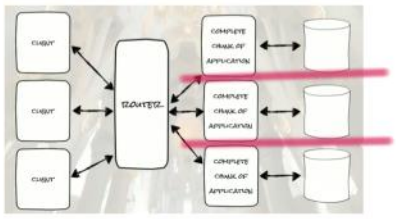


(3) sharded application

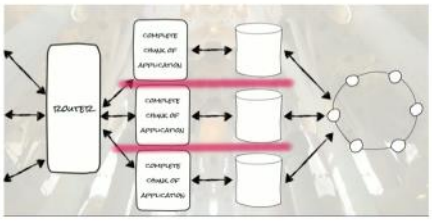
1) как обычно



2) sharded



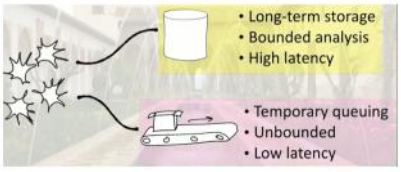
3) sharded + overall data view



минус в том что требуется склеивать все шарды для того чтобы обеспечить общее представление всех данных

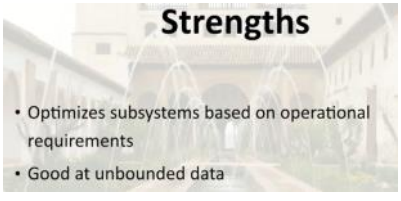
(5) lambda architecture (unbounded immutable data)

lambda



lambda + overall data view

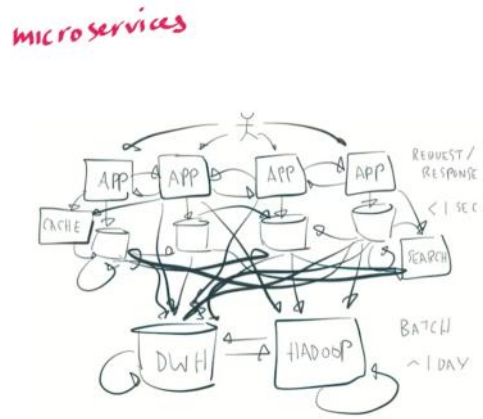
самое главное преимущество в том что можно оптимизировать подсистемочки (независимо друг от друга)



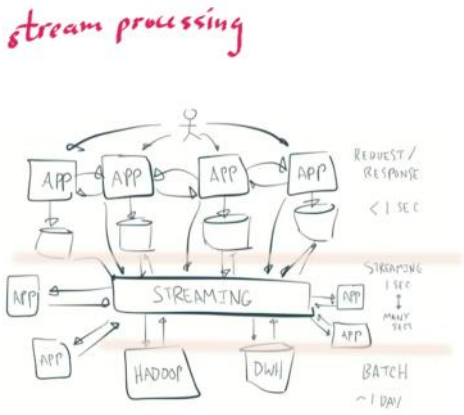
минус в том что с помощью ламбда не построить абсолютно все части приложения (она не везде подходит)

(6) stream processing architecture (kafka based)

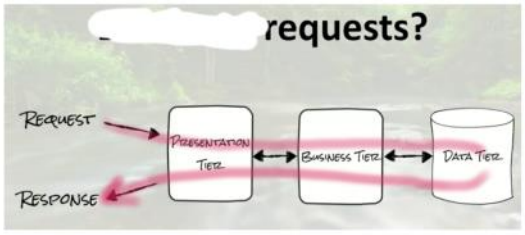
- тоже формально относится к лямбда архитектуре

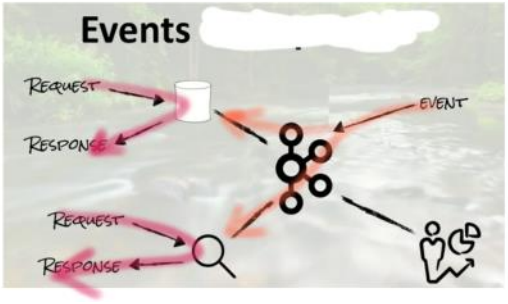


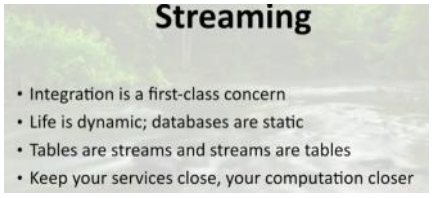
обычная трехзвенка

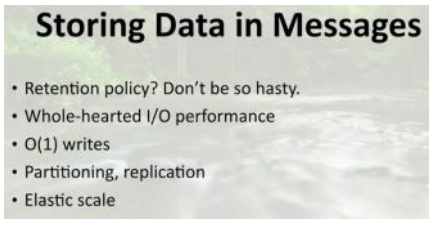


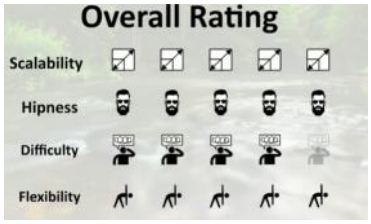
stream processing











* эволюция архитектур

3 февраля 2021 г. 12:02

три эпохи работы с данными

(1) The first wave of big data was "data at rest." We stored massive amounts in Hadoop Distributed File System (HDFS) or similar, and then had offline batch processes crunching the data over night, often with hours of latency.
(2) in the second wave, we saw that the need to react in real time to the "data in motion"—to capture the live data, process it, and feed the
result back into the running system within seconds and sometimes even subsecond response time—had become increasingly important. (Lambda Architecture)
(3) The third wave—distributed streaming—is the one that is most interesting to microservices-based architectures (Flink, Spark Streaming, and Google Cloud Dataflow.)

- когда потребности обработки данных стали совершенно явственными, была разработана новая стратегия — микропакетирование (microbatching). Как понятно из названия, оно представляет собой просто пакетную обработку, но с меньшими объемами данных. Благодаря снижению размера пакета микропакетирование позволяет иногда получать результаты быстрее, но это все равно пакетная обработка, хотя и с более короткими промежутками времени. Это не настоящая обработка по событиям.

- *The classic era*—The pre-big data, pre-SaaS era of operational systems and batch-loaded data warehouses
- *The hybrid era*—Today's hodgepodge of different systems and approaches
- *The unified era*—An emerging architecture, enabled by processing continuous event streams in a unified log

сравнение подходов: спагетти, SOA, unified log

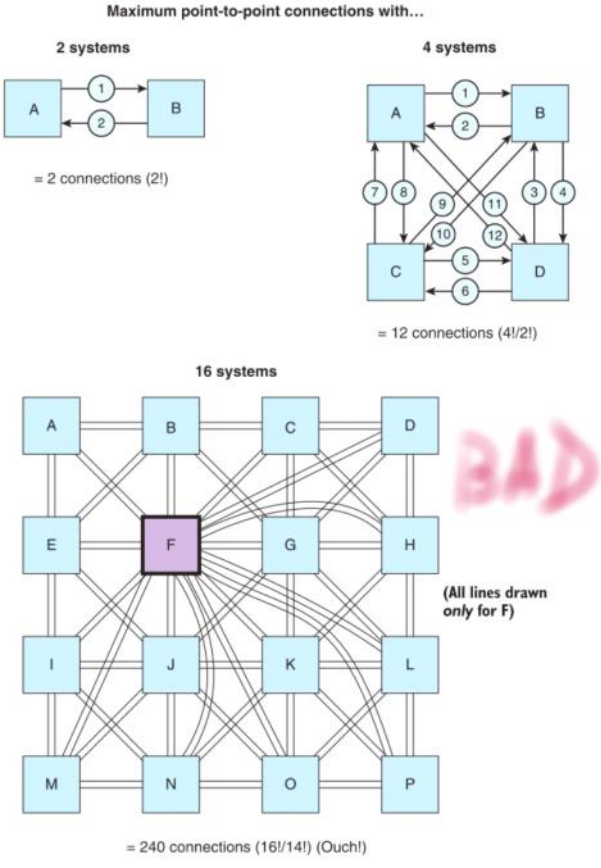


Figure 1.8 The maximum number of point-to-point connections possibly required between 2, 4, and 16 software systems is 2, 12, and 240 connections, respectively. Adding systems grows the number of point-to-point connections quadratically.

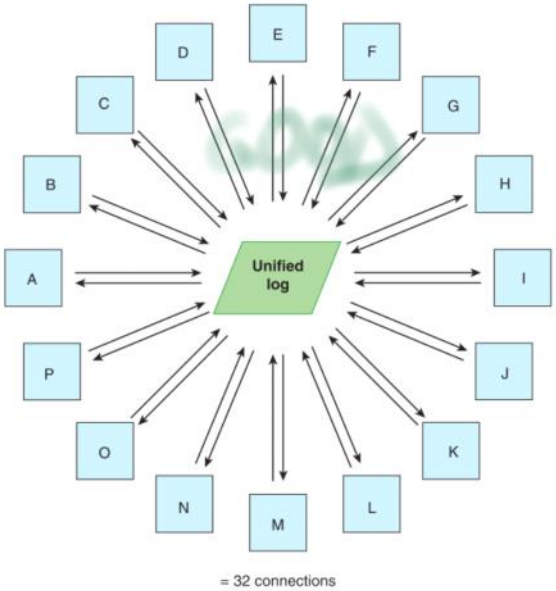


Figure 1.11 Our unified log acts as the glue between all of our software systems. In place of the proliferation of point-to-point connections seen prior, we now have systems reading and writing to the unified log. Conceptually, we now have a maximum of 32 unidirectional connections, compared to 240 for a point-to-point approach.

Фундаментальное различие между фреймворками пакетной обработки и фреймворками потоковой обработки связано с тем, как они получают данные.

- Платформы пакетной обработки ожидают, что они будут работать с завершенным набором записей (terminated set of records) , в отличие от безграничного(unbounded)потока событий (или потоков),

- историческое отличие состоит в том, что структуры пакетной обработки использовались с гораздо более широким спектром данных, чем структуры обработки потоковой передачи.

Например подсчет слов в корпусе англоязычных документов (полуструктурированные данные). В отличие от этого, фреймворки потоковой обработки больше ориентированы на хорошо структурированные данные потока событий,

работает с конечными кусками информации

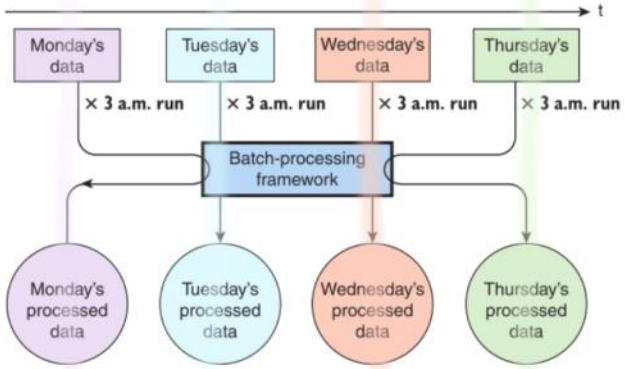


Figure 7.9 A batch processing framework has processed four distinct batches of events, each belonging to a different day of the week. The batch processing framework runs at 3 a.m. daily, ingests the data for the prior day from storage, and writes its outputs back to storage at the end of its run.

безграничный поток

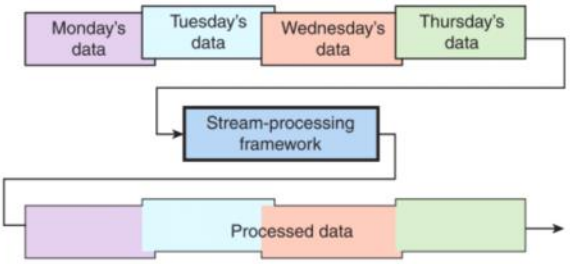


Figure 7.10 A stream processing framework doesn't distinguish any breaks in the incoming event stream. Monday through Thursday's data exists as one unbounded stream, likely with overlap due to late-arriving events.

1 The classic era

- An internal *local loop* for near-real-time data processing
- Its own data silo
- Where necessary, point-to-point connections to peer systems (for example, via APIs or feed import/exports)

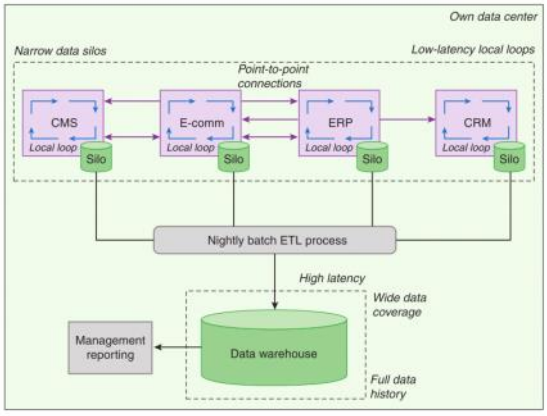


Figure 1.7 This retailer has four transactional systems, each with its own data silo. These systems are connected to each other as necessary with point-to-point connections. A nightly batch ETL process extracts data out of the data silos, *transforms* it for reporting purposes, and then *loads* it into a data warehouse. Management reports are then based on the contents of the data warehouse.

- *High latency for reporting*—The time span between an event occurring and that event appearing in management reporting is counted in hours (potentially even days), not seconds
- *Point-to-point spaghetti*—Extra transactional systems mean even more point-to-point connections, as illustrated in figure 1.8. This point-to-point spaghetti is expensive to build and maintain and increases the overall fragility of the system.
- *Schema woes*—Classic data warehousing assumes that each business has an intrinsic data model that can be mined from the state stored in its transactional systems. This is a highly flawed assumption, as we explore in chapter 5.

2 The hybrid era (batch processing, microbatch processing)

- у них есть сильные локальные петли и разрозненные хранилища данных, но есть также попытки «регистрировать все» с помощью Hadoop и / или системного мониторинга
- Как правило, это сочетание обработки в режиме, близком к реальному времени, для узких вариантов использования аналитики, таких как рекомендации по продукту, плюс отдельные усилия по пакетной обработке в Hadoop, а также классическое хранилище данных

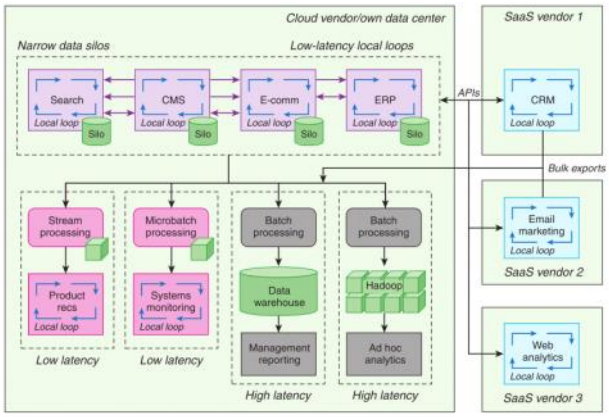


Figure 1.9 Compared to the classic era, our retailer has now added external SaaS dependencies; Hadoop as a new high-latency, "log everything" platform; and new low-latency data pipelines for use cases such as systems monitoring and product recommendations.

- *No single version of the truth*—Data is now warehoused in multiple places, depending on the data volumes and the analytics latency required. There is no system that has 100% visibility.
- *Decisioning has become fragmented*—The number of local systems loops, each operating on siloed data, has grown since the classic era. These loops represent a highly fragmented approach to making near-real-time decisions from data.
- *Point-to-point connections have proliferated*—As the number of systems has grown, the number of point-to-point connections has exploded. Many of these connections are fragile or incomplete; getting sufficiently granular and timely data out of external SaaS systems is particularly challenging.
- *Analytics can have low latency or wide data coverage, but not both*—When stream processing is selected for low latency, it becomes effectively another local processing loop. The warehouses aim for much wider data coverage, but at the cost of duplication of data and high latency.

3 The unified era

- Ключевым нововведением с точки зрения бизнеса является создание единого журнала в основе всей нашей деятельности по сбору и обработке данных.
- Единый журнал является конкатенирующим только журнал , к которому мы пишем все события , генерируемые наших приложения.

- Can be read from at low latency.
- Is readable by multiple applications simultaneously, with different applications able to consume from the log at their own pace.
- Holds only a rolling window of events—probably a week or a month's worth. But we can archive the historic log data in the Hadoop Distributed File System (HDFS) or Amazon Simple Storage Service (S3).

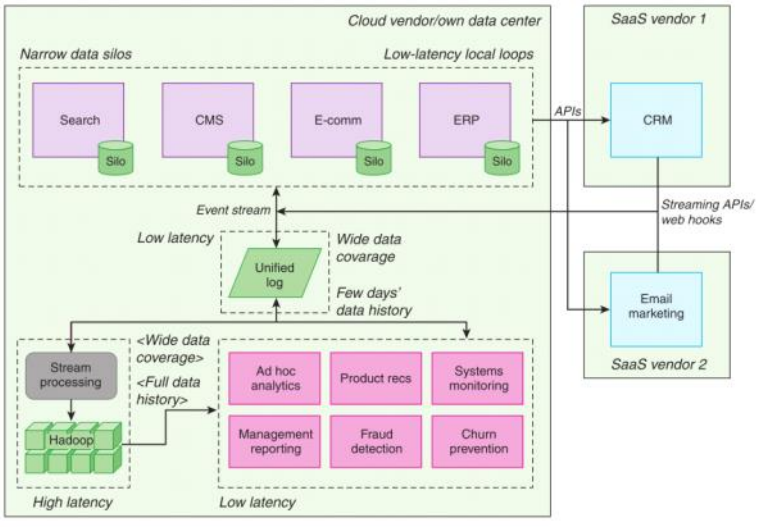


Figure 1.10 Our retailer has rearchitected around a unified log and a longer-term archive of events in Hadoop. The data architecture is now much simpler, with far fewer point-to-point connections, and all of our analytics and decision-making systems now working off a single version of the truth.

Все программные системы могут и должны записывать свои отдельные непрерывные потоки событий в единый журнал. Даже сторонние поставщики SaaS могут отправлять события через веб-перехватчики и потоковые API.

Если не требуется очень низкая задержка или гарантии транзакций, программные системы должны взаимодействовать друг с другом несвязанным способом через унифицированный журнал, а не через соединения точка-точка.

-

- *We have a single version of the truth.* Together, the unified log plus Hadoop archive represent our single version of the truth. They contain exactly the same data—our event stream—but they have different time windows of data.
- *The single version of the truth is upstream from the data warehouse.* In the classic era, the data warehouse provided the single version of the truth, making all reports generated from it consistent. In the unified era, the log provides the single version of the truth; as a result, operational systems (for example, recommendation and ad-targeting systems) compute on the same truth as analysts producing management reports.
- *Point-to-point connections have largely been unravelled.* In their place, applications can append to the unified log, and other applications can read their writes. This is illustrated in figure 1.11.
- *Local loops have been unbundled.* In place of local silos, applications can collaborate on near-real-time decision-making via the unified log.

Some schema technologies

3 февраля 2021 г. 22:02

- *Multiple schema languages* (LNG)—Some schema technologies provide multiple ways in which you can express the data types for your business entities. For example, you may be able to write your schemas declaratively in JSON, or there may some form of interface description language (IDL) with a more C-like or Java-like syntax.
- *Validation rules* (VLD)—Some schema technologies go further than data types, by letting you express validation rules (sometimes called *contracts*) on the event's properties. For example, you might express that a longitude is not just a floating-point number, but a floating-point number that can't be more than 180 or less than –180, and that latitude must be no more than 90 or less than –90.
- *Code generation* (GEN)—Whatever syntax the schema is expressed in, we are likely to want to also interact with events represented by the schema in code we write (for example, our stream processing apps). To facilitate this, schema technologies often support code generation, which will generate idiomatic classes or records in your preferred language from the schema.
- *Multiple encodings* (ENC)—Some schema technologies support multiple encodings of the data, often a compact binary format and a human-readable format (perhaps JSON-based).
- *Schema evolution* (EVO)—The properties in our Plum events will likely evolve over time. Some of the more sophisticated schema technologies have built-in support for schema evolution, which makes it easier to consume different versions of the same schema.
- *Remote procedure calls* (RPC)—This is not a feature that we need, but some data serialization systems also come with a mechanism for building remote procedure calls, distributed functions that use data types for expressing the functions' arguments and return values.

Table 6.1 Examples of schema technologies

Schema tech	LNG	VLD	GEN	ENC	EVO	RPC
Apache Avro	JSON, IDL	No	Yes	Binary, JSON	Yes	Yes
Apache Thrift	IDL	No	Yes	Five encodings	Yes	Yes
JSON Schema	JSON	Yes	No	JSON	No	No
Protocol buffers	IDL	No	Yes	Binary, JSON	No	Yes (gRPC)

Stream processing frameworks exhibit some or all of the following capabilities:

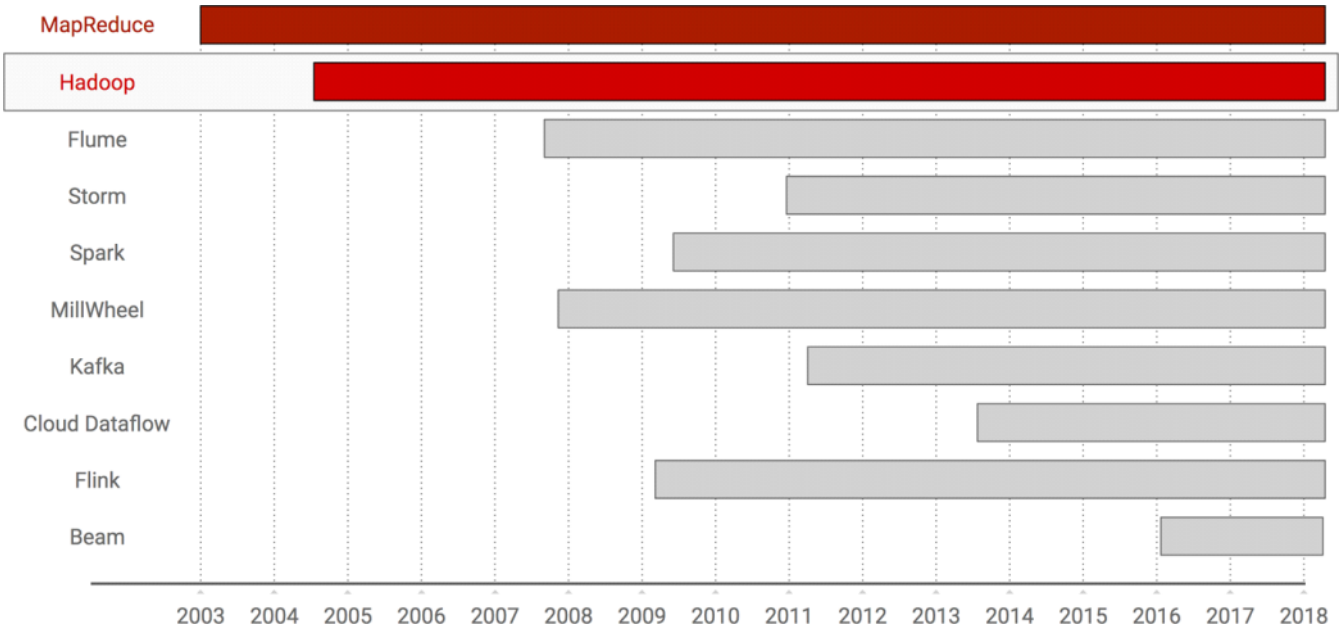
- *State management*—State is a key building block for processing multiple events at a time. Stream processing frameworks give you the option of storing state in some or all of the following: in-memory, in a local filesystem, or in a dedicated key-value store such as RocksDB or Redis.
- *Stream windowing*—As described previously, a stream processing framework provides one or more ways of expressing a *bounded window* for the event processing. This is typically time-based, although sometimes it can be based on a count of events instead.
- *Delivery guarantees*—All stream processing frameworks pessimistically track their progress, to ensure that every event is processed *at least once*. Some of these systems go further, adding in transactional guarantees to ensure that each event is processed only *exactly once*.
- *Task distribution*—A high-volume event stream consists of multiple Kafka topics or Amazon Kinesis streams, and requires multiple instances of the job, or tasks, to process it. Most stream processing frameworks are designed to be run on a sophisticated third-party scheduler such as Apache Mesos or Apache Hadoop YARN; some stream processing frameworks are also *embeddable*, meaning that they can be added as a library to a regular application (requiring bespoke scheduling).
- *Fault tolerance*—Failures happen regularly in the kind of large-scale distributed systems required to process high-volume event streams, and most frameworks have built-in mechanisms to automatically recover from these failures. Fault tolerance typically involves a distributed backup of either the incoming events or the generated state.

Table 5.1 A nonexhaustive list of stream processing frameworks

Capability	Storm	Samza	Spark Streaming	Kafka Streams	Flink
State management	In-memory, Redis	In-memory, RocksDB	In-memory, filesystem	In-memory, RocksDB	In-memory, filesystem, RocksDB
Stream windowing	Time-, count-based	Time-based	Time-based (microbatch)	Time-based	Time-, count-based
Delivery guarantees	At least once, exactly once (Trident)	At least once	At least once, exactly once	At least once, exactly once	Exactly once
Task distribution	YARN or Mesos	YARN or embeddable	YARN or Mesos	Embeddable	YARN or Mesos
Fault tolerance	Record acks	Local, distributed snapshots	Checkpoints	Local, distributed snapshots	Distributed snapshots

Table 7.3 Examples of distributed batch-processing frameworks

Framework	Started	Creator	Description
Disco	2008	Nokia Research Center	MapReduce framework written in Erlang. Has its own filesystem, DDFS.
Apache Flink	2009	TU Berlin	Formerly known as Project Stratosphere, Flink is a streaming dataflow engine with a DataSet API for batch processing. Write jobs in Scala, Java, or Python.
Apache Hadoop	2008	Yahoo!	Software framework written in Java for distributed processing (Hadoop MapReduce) and distributed storage (HDFS).
Apache Spark	2009	UC Berkeley AMPLab	Large-scale data processing, supporting cyclic data flow and optimized for memory use. Write jobs in Scala, Java, or Python.

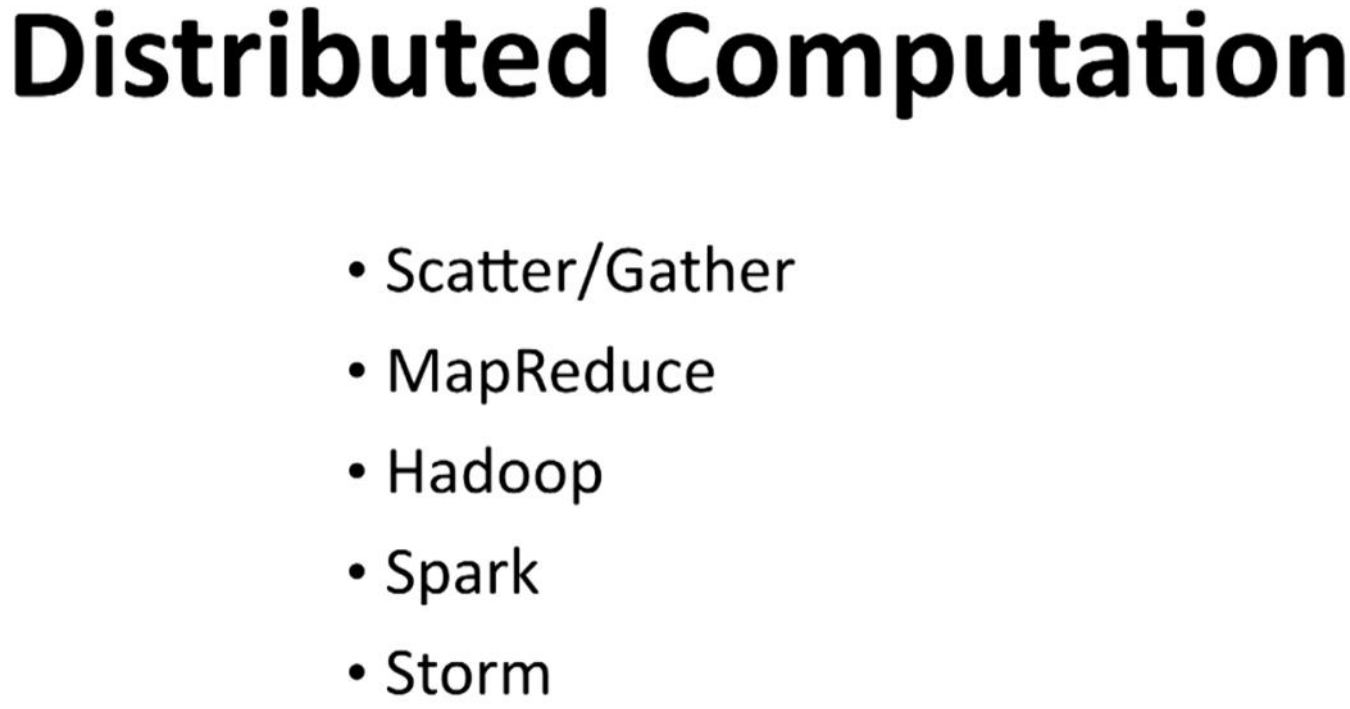


SCATTER/GATHER

7 марта 2021 г. 20:34

паттерн scatter/gather -как самый базовый примитивный шаблон распределенных вычислений

см * APACHE SPARK STREAMING



google MapReduce

8 февраля 2021 г. 19:26

onenote:///C:\Users\trans\Dsync\vova_from_onenote\tf_algonote_v1\АРХИТЕКТУРА%20КОМПАНИЙ\google.one#поиск%20reyrn&section-id={3A83F109-029C-458E-9276-68837C57517A}&page-id={C917CA36-9CA7-4F0F-9574-A69DB870AEE0}&end

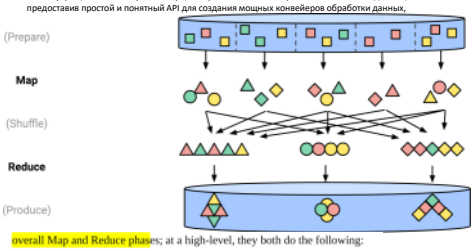
- Предоставляя простой набор абстракций для обработки данных поверх надежного и масштабируемого механизма выполнения, MapReduce позволил инженерам по обработке данных сосредоточиться на бизнес-логике своих потребностей в обработке данных, а не на сложных деталях построения распределенных систем, устойчивых к режимам отказа. товарное оборудование.

2) MapReduce - применение этих простых принципов в колоссальных масштабах на множестве машин.

https://cloud.google.com/blog/products/gcp/history-of-massive-scale-sorting-experiments-at-google

Но для работы алгоритма PageRank в масштабах всего контента Интернета необходим был другой подход — традиционные подходы к работе с данными были слишком медленными. Чтобы выжить и развиваться, Google необходимо было быстро (где «быстро» — понятие относительное) индексировать весь этот контент и предоставить аудитории качественные результаты. И компания Google разработала еще один революционный подход для обработки
всех этих данных — парадигму MapReduce («отображение — свертка»). Парадигма MapReduce не только дала Google возможность выполнить всю необходимую работу, но и нечаянно породила целую новую отрасль вычислений.

- С точки зрения стриминговых систем, основные выводы, которые я хочу оставить вам для MapReduce, - это простота и масштабируемость . MapReduce предпринял первые смелые шаги к укрощению непокорного зверя, который занимается обработкой массивных данных, предоставив простой и понятный API для создания мощных конвейеров обработки данных,

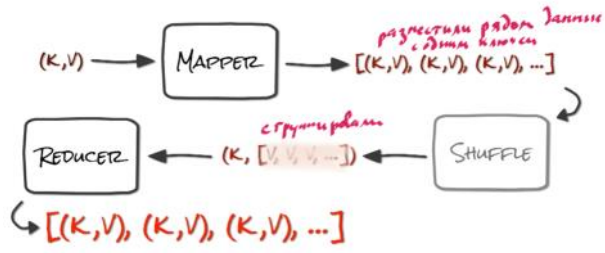


overall Map and Reduce phases; at a high-level, they both do the following:

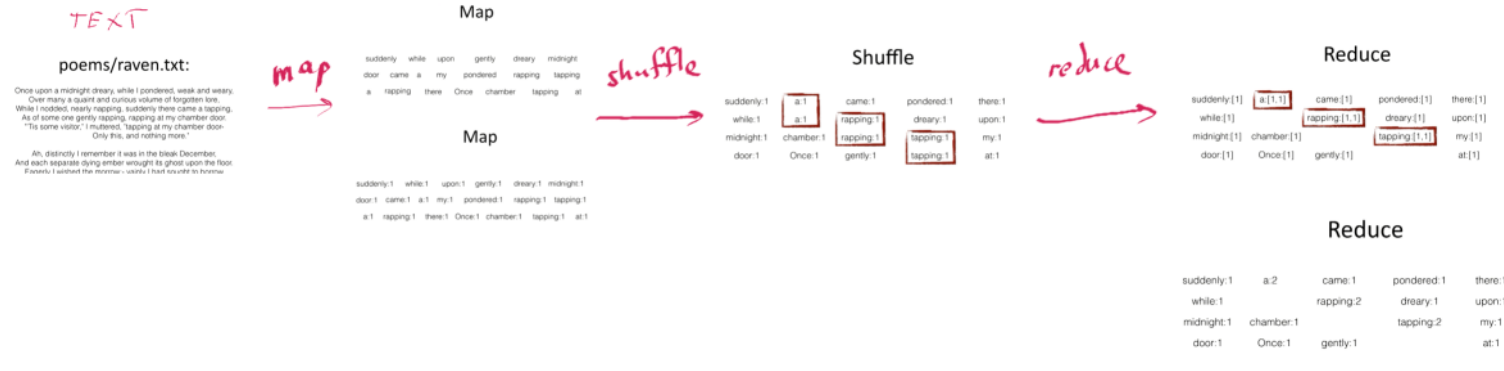
- Convert a table to a stream
- Apply a user transformation to that stream to yield another stream
- Group that stream into a table

shuffle - неявно делает фреймворк

алгоритм должен был бы называться



Пример



Пакетная обработка идеально подходит для алгоритмов, подобных PageRank

- Но использование Hadoop/MapReduce, по сути, пакетный процесс, так что приходится собирать большие объемы данных, обрабатывать их, а затем сохранять результаты для дальнейшего применения. Пакетная обработка идеально подходит для алгоритмов, подобных PageRank, поскольку все равно нельзя принимать решения о ценности ресурсов в масштабах всего Интернета на основе наблюдения за переходами пользователей по ссылкам в режиме реального времени.

инженеры гугл заметили три вещи

-
- Я думаю, можно с уверенностью сказать, что крупномасштабная обработка данных в том виде, в каком мы все ее знаем сегодня, началась с MapReduce еще в 2003 году. 2 В то время инженеры Google создавали всевозможные индивидуальные системы для решения проблем обработки данных в масштаб всемирной паутины. 3 Как они сделали это, они заметили три вещи:
- После решения всех трех задач в тандеме в нескольких сценариях использования они начали замечать некоторые сходства между созданными ими пользовательскими системами. И они пришли к выводу, что, если бы они могли создать фреймворк, который позаботился бы о последних двух проблемах (масштабируемость и отказоустойчивость), было бы намного проще сосредоточиться на первом вопросе. Так родился MapReduce.
- Основная идея MapReduce заключалась в том, чтобы предоставить простой API обработки данных, сосредоточенный на двух хорошо понятных операциях из области функционального программирования: отображение и сокращение (рис. 10-3). Конвейеры, построенные с помощью этого API, затем будут выполняться на платформе распределенных систем,

которая позаботится обо всех неприятных вещах, связанных с масштабируемостью и отказоустойчивостью, которые оживляют сердца заядлых инженеров распределенных систем и сокрушают души всех остальных, простых смертных.

Data processing is hard

As the data scientists and engineers among us well know, you can build a career out of just focusing on the best ways to extract useful insights from raw data.

Scalability is hard

Extracting useful insights over massive-scale data is even more difficult yet.

Fault-tolerance is hard

Extracting useful insights from massive-scale data in a fault-tolerant, correct way on commodity hardware is brutal.

The three main problems that the MapReduce paper solved are:
1. Parallelization — how to parallelize the computation
2. Distribution — how to distribute the data
3. Fault-tolerance — how to handle program failure

Отказоустойчивость MapReduce

- The master pings every worker periodically. If no response is received from a worker in a certain amount of time, the master marks the worker as failed. Any map tasks, in-progress or completed by the failed worker are reset back to their initial, idle state, and therefore become eligible for scheduling on other workers.

MapReduce рассчитана на частые непредвиденные прерывания задач

- : не потому, что аппаратное обеспечение ненадежно, а для возможности произвольного прерывания процессов с целью лучше использовать ресурсы в вычислительном кластере.

команды разработчиков должны оплачивать ресурсы, которые они используют, и процессы с более высоким приоритетом стоят дороже

- Чтобы понять причины экономного использования памяти и восстановления на уровне операций в MapReduce, полезно взглянуть на ту среду, для которой изначально создавалась система MapReduce.
- У Google есть многофункциональные центры обработки и хранения информации, где операционные онлайновые сервисы и автономные пакетные задания выполняются на одних и тех же машинах.
- Каждой задаче предоставляются определенные ресурсы (ядра процессора, оперативная память, дисковое пространство и т. п.), активизируемые с помощью контейнеров.
- У каждой задачи также есть приоритет: если задаче с более высоким приоритетом нужно больше ресурсов, то задачи с более низким приоритетом, выполняемые на том же компьютере, могут быть прерваны (выгружены), чтобы освободить ресурсы.
- Вдобавок приоритет определяет цену вычислительных ресурсов: команды разработчиков должны оплачивать ресурсы, которые они используют, и процессы с более высоким приоритетом стоят дороже
- Эта архитектура допускает чрезмерную нагрузку на непроизводственные (низкоприоритетные) вычислительные ресурсы, поскольку система при необходимости может их освободить
- В свою очередь, избыточные ресурсы позволяют лучше использовать вычислительные мощности и повысить их эффективность по сравнению с системами, где разделяются производственные и непроизводственные задачи

задачи MapReduce выполняются с низким приоритетом

- Однако, поскольку задачи MapReduce выполняются с низким приоритетом, есть риск, что они будут прерваны в любое время, так как их ресурсы потребуются для процесса с более высоким приоритетом. Пакетные задачи эффективно «собирают объедки под столом», задействуя любые вычислительные ресурсы, остающиеся после процессов с высоким приоритетом

вероятность того, что по крайней мере одна из цепочек задач будет прервана до завершения, составляет более 50 %

- В Google вероятность того, что задача MapReduce, работающая в течение часа, будет прервана с целью освободить ресурсы для процесса с более высоким приоритетом, составляет примерно 5 %. Это более чем на порядок выше вероятности сбоя из-за проблем с оборудованием, перезагрузки компьютера или по другим причинам [59]. При такой возможности прерывания, если задача состоит из 100 операций, каждая из которых выполняется в течение 10 минут, вероятность того, что по крайней мере одна из них будет прервана до завершения, составляет более 50 %

Одна из ключевых идей MapReduce заключалась в том, что мы вообще должны заставлять перемещать данные для их обработки.

- Вместо этого программа отправляется туда, где находятся данные. Это ключевое отличие от традиционных систем хранилищ данных и реляционных баз данных. Просто слишком много данных, чтобы их можно было перемещать.

MapReduce фактически построен на файловой системе Google, GFS, которая сама решает проблемы масштабируемости и отказоустойчивости для определенного подмножества общей проблемы.
рассматривать отказ как ежечасное типовое событие (а не как проблему)

- когда Google только начинал свою деятельность, они столкнулись с проблемой отказа жесткого диска в своих центрах обработки данных. Поскольку их основным бизнесом был (и остается) «данные», они легко обосновали решение о постепенной замене вышедших из строя недорогих дисков более дорогими и первоклассными. По мере того как компания росла в геометрической прогрессии, росло и общее количество дисков, и вскоре они насчитали миллионы жестких дисков. Это решение продлило срок службы диска, если рассматривать каждый диск отдельно, но в таком большом пуле оборудования все равно неизбежно выходили из строя диски, почти ежечасно. Это означало, что им все еще приходилось иметь дело с той же проблемой, поэтому они постепенно вернулись к обычным, стандартным жестким дискам и вместо этого решили решить проблему, рассматривая отказ компонентов не как исключение, а как обычное явление.

	IBM HD	Commodity HD	USB stick
Year	1987	2015	2015
Weight	120 Kg	635 g	5 g
Capacity	3.78 GB	3 TB	128 GB
Price	$128 000	$105.53	$35.99
Price per GB	$33 862.43	$0.034	$0.281

Cost of memory over time

To данным наоборот: не данные подтягиваем к вычислениям, а вычисления к данным

- All computation in two functions: Map and Reduce
- Keep data (mostly) where it is
- Move compute to data

За годы, прошедшие с момента первого появления MapReduce, механизмы распределенной пакетной обработки стали совершеннее.

- В настоящее время инфраструктура достаточно надежна, чтобы хранить и обрабатывать многие петабайты данных в кластерах, насчитывающих более 10 000 машин.
- Поскольку задача создать физически работающий пакетный процесс такого масштаба более или менее решена, внимание переключилось на другие области: совершенствование модели программирования, повышение эффективности обработки и расширение набора задач, решаемых с помощью этих технологий.

языки более высокого уровня и API, такие как Hive, Pig, Cascading и Crunch, стали

- В настоящее время инфраструктура достаточно надежна, чтобы хранить и обрабатывать многие петабайты данных в кластерах, насчитывающих более 10 000 машин.
- Поскольку задача создать физически работающий пакетный процесс такого масштаба более или менее решена, внимание переключилось на другие области: совершенствование модели программирования, повышение эффективности обработки и расширение набора задач, решаемых с помощью этих технологий.

языки более высокого уровня и API, такие как Hive, Pig, Cascading и Crunch, стали популярными

- поскольку программировать задачи вручную в MapReduce — довольно трудоемкий процесс
- после появления Tez эти языки высокого уровня получили дополнительное преимущество: возможность перейти к новому механизму обработки потока данных без необходимости переписывать код задачи.
- В Spark и Flink также имеются собственные потоковые API высокого уровня, которые во многом черпают вдохновение в FlumeJava

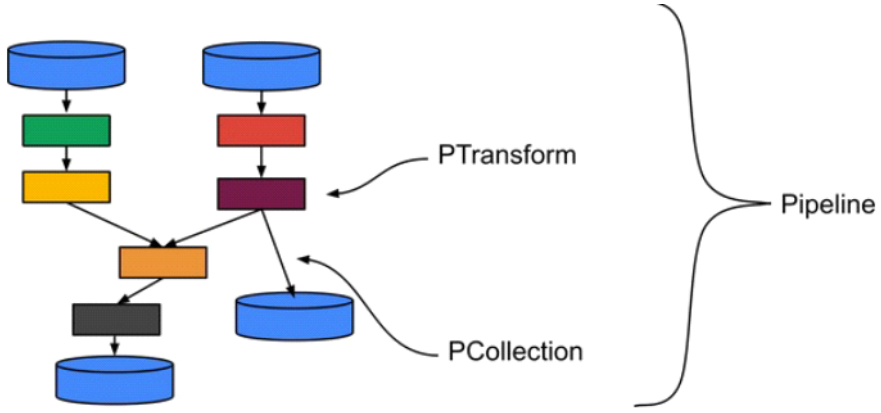
* Google Flume (это не apache flume)

8 февраля 2021 г. 20:16

официальном преемнике MapReduce в Google: Flume

- ([рис. 10-8], иногда также называемом FlumeJava в связи с исходной версией системы Java, и не путать с Apache Flume, который это совершенно другой зверь, который случайно носит одно и то же имя).
- Он был мотивирован желанием устранить некоторые из внутренних недостатков MapReduce, которые стали очевидны в первые несколько лет его успеха. Многие из этих недостатков связаны с жесткой структурой Map → Shuffle → Reduce в MapReduce;
- https://storage.googleapis.com/pub-tools-public-publication-data/pdf/35650.pdf
- https://cloud.google.com/blog/products/gcp/no-shard-left-behind-dynamic-work-rebalancing-in-google-cloud-dataflow
- Объединив высокоуровневое понятие логических операций конвейера с интеллектуальным оптимизатором, Flume позволил писать чистые и обслуживаемые конвейеры, чьи возможности выходили за рамки Map → Shuffle → Reduce ограничения MapReduce, не жертвуя при этом какой-либо производительностью, достигнутой ранее за счет искажения. логический конвейер с помощью вручную настроенных оптимизаций.

Flume решил эти проблемы, предоставив составной высокоуровневый API для описания конвейеров обработки данных, в основном основанный на тех же концепциях PCollection и PTransform, что и в Beam



самое важное во flume - это то что он может выполнять автоматическую оптимизацию

- главное, что нужно вынести из Flume в этом разделе, - это введение понятия высокоуровневых конвейеров , которое позволяет автоматически оптимизировать четко написанные логические конвейеры. Это позволило создавать гораздо более крупные и сложные конвейеры без необходимости ручной оркестровки или оптимизации, при этом код для этих конвейеров оставался логичным и понятным.

```
PCollection<KV<String, Long>> sums =
  IO.read(...)
    .ParDo(new KeyByUser())
    .Count();
```

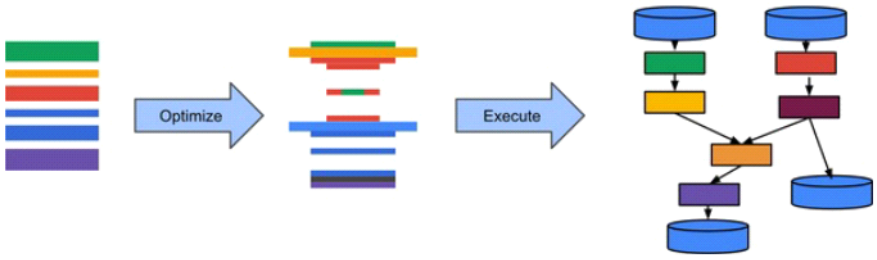


Figure 10-10. Optimization from a logical pipeline to a physical execution plan

примеры fusion оптимизаций

- Возможно, наиболее важным примером автоматической оптимизации, которую может выполнять Flume, является слияние), в котором два логически независимых этапа могут выполняться в одном задании либо последовательно (слияние потребителя и производителя), либо параллельно (слияние братьев и сестер), как показано на рисунке
- Объединение двух этапов вместе исключает сериализацию / десериализацию и сетевые затраты, которые могут быть значительными в конвейерах, обрабатывающих большие объемы данных.

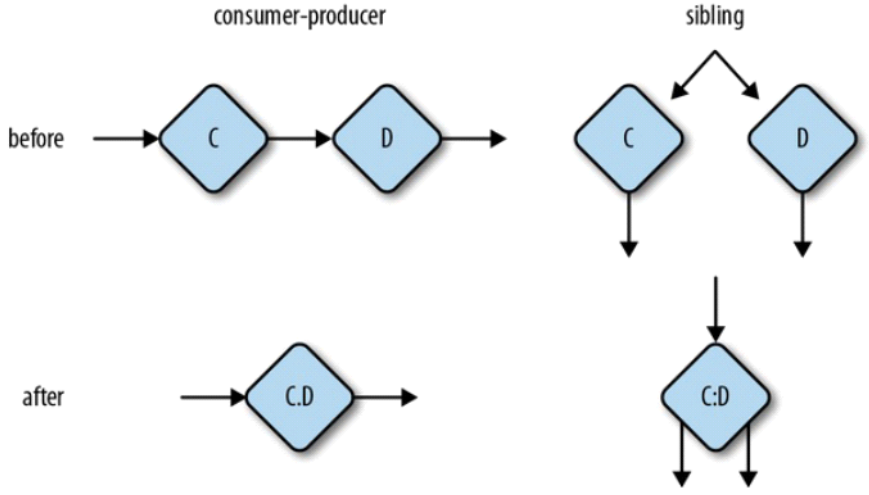


Figure 10-11. Fusion optimizations combine successive or parallel operations together into the same physical operation

пример combiner lifting оптимизации

- пошаговом комбинировании. Подъем комбайнера - это просто автоматическое применение многоуровневой логики комбинирования
- операция комбинирования (например, суммирование), которая логически происходит после того, как операция группирования частично переносится на стадию, предшествующую группировке по ключам (которая по определению требуется поездка по сети для перетасовки данных), чтобы можно было выполнить частичное объединение до того, как произойдет группировка.
- В случае очень горячих клавиш это может значительно уменьшить объем данных, перетасовываемых по сети, а также более плавно распределить нагрузку вычисления конечного агрегата на несколько машин.

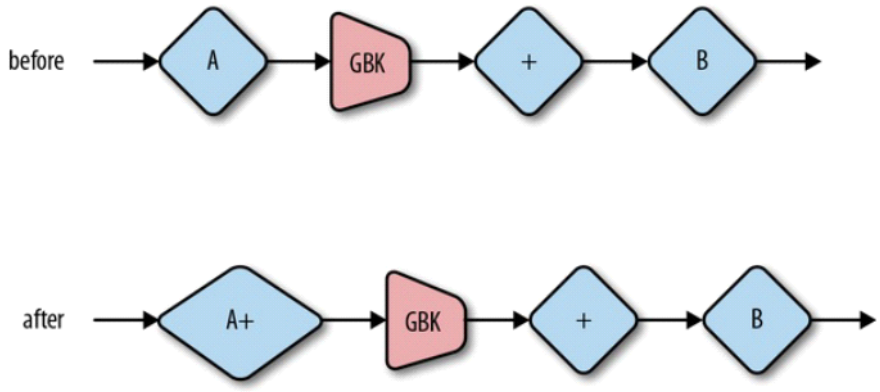


Figure 10-12. Combiner lifting applies partial aggregation on the sender side of a group-by-key operation before completing aggregation on the consumer side

Hadoop (Nutch crawler)

8 февраля 2021 г. 19:26

https://medium.com/@markobonaci/the-history-of-hadoop-68984a11704

см также * распределенная файловая система

HDFS

- Files and directories
- Metadata managed by a replicated master
- Files stored in large, immutable, replicated blocks

- Hadoop появился в 2005 году, когда Дуг Каттинг и Майк Кафарелла решили, что идеи из статьи о MapReduce - именно то, что им было нужно при создании распределенной версии своего веб-краулера Nutch.
- Они уже создали свою собственную версию распределенной файловой системы Google (первоначально называвшуюся NDFS для Nutch Distributed File System, позже переименованную в HDFS или Hadoop Distributed File System), поэтому естественным следующим шагом было добавление слоя MapReduce поверх этой статьи. был опубликован. Они назвали этот слой Hadoop.
- Создав платформу с открытым исходным кодом на идеях MapReduce, Hadoop создал процветающую экосистему, которая расширилась далеко за пределы ее возможностей. прародитель и позволил процветать множеству новых идей.

экосистема с открытым исходным кодом, которая процветала вокруг Hadoop, оказала на отрасль в целом.

- Ключевое различие между Hadoop и MapReduce заключалось в том, что Cutting и Cafarella позаботились о том, чтобы исходный код Hadoop был передан остальному миру, открыв его (вместе с исходным кодом для HDFS) как часть того, что в конечном итоге станет проектом Apache Hadoop.
- Главный момент, который я хочу, чтобы вы вынесли из этого раздела, - это огромное влияние, которое экосистема с открытым исходным кодом, которая процветала вокруг Hadoop, оказала на отрасль в целом. Создав открытое сообщество, в котором инженеры могли улучшать и расширять идеи из тех ранних статей GFS и MapReduce, родилась процветающая экосистема, дающая десятки полезных инструментов, таких как Pig, Hive, HBase, Crunch и т. Д. Эта открытость была ключом к инкубации разнообразия идей, существующих сейчас в нашей отрасли, и именно поэтому я рассматриваю экосистему с открытым исходным кодом Hadoop как ее самый важный вклад в мир потоковых систем, какими мы их знаем сегодня.

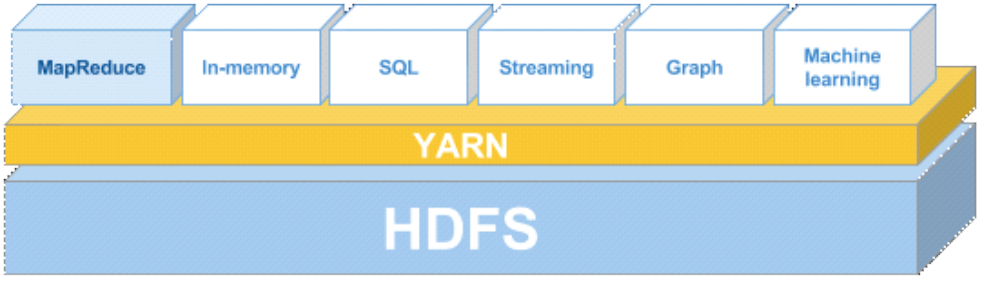
Hadoop представляет собой нечто похожее на распределенную версию Unix.

- На самом абстрактном уровне базы данных Hadoop и операционные системы выполняют одни и те же функции: хранят данные, позволяют их обрабатывать и запрашивать [16].
- В базе данные хранятся в виде записей, построенных по определенной информационной модели (строки в таблицах, документы, вершины графа и т. п.).
- В файловой системе они хранятся в файлах, но по своей сути то и другое — системы «управления информацией»

- Конечно, на практике здесь есть много различий. Например, многие файловые системы не очень хорошо справляются с каталогом, содержащим 10 млн небольших файлов, тогда как для базы данных такое количество маленьких записей — совершенно нормальная, ничем не примечательная ситуация

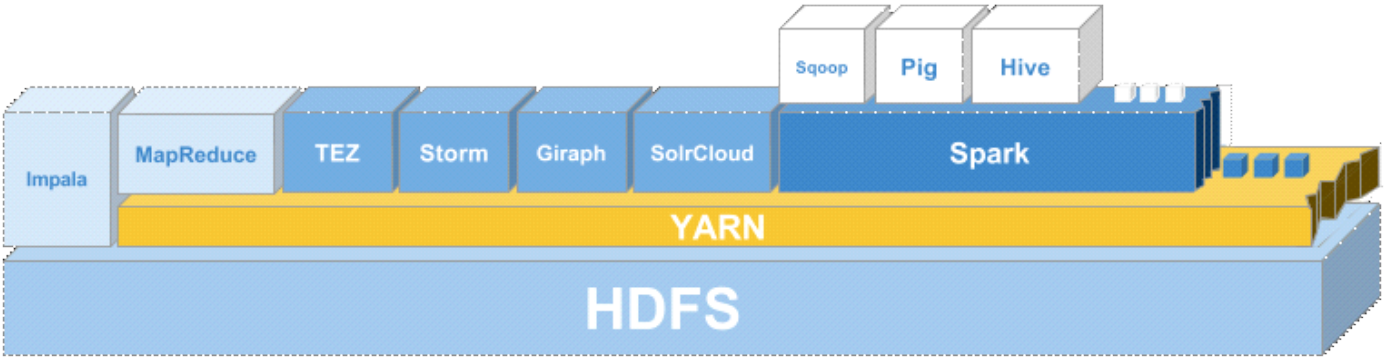
YARN

- Корень всех проблем был в том, что у MapReduce было слишком много обязанностей. Он практически отвечал за все, что выше уровня HDFS, назначая ресурсы кластера и управляя выполнением заданий (система), выполняя обработку данных (движок) и взаимодействуя с клиентами (API). Следовательно, у фреймворков более высокого уровня не было другого выбора, кроме как строить поверх MapReduce
- Хотя MapReduce выполнила свою миссию по обработке ранее непреодолимых объемов данных, стало очевидно, что необходима более общая и более гибкая платформа на основе HDFS.
- In order to generalize processing capability, the resource management, workflow management and fault-tolerance components were removed from MapReduce, a user-facing framework and transferred into YARN, effectively decoupling cluster operations from the data pipeline.



Он демократизировал область фреймворков приложений, стимулировал инновации во всей экосистеме и породил множество новых, специально созданных фреймворков.

- MapReduce был изменен (полностью обратно совместимым), так что теперь он работает поверх YARN как одна из многих различных платформ приложений.

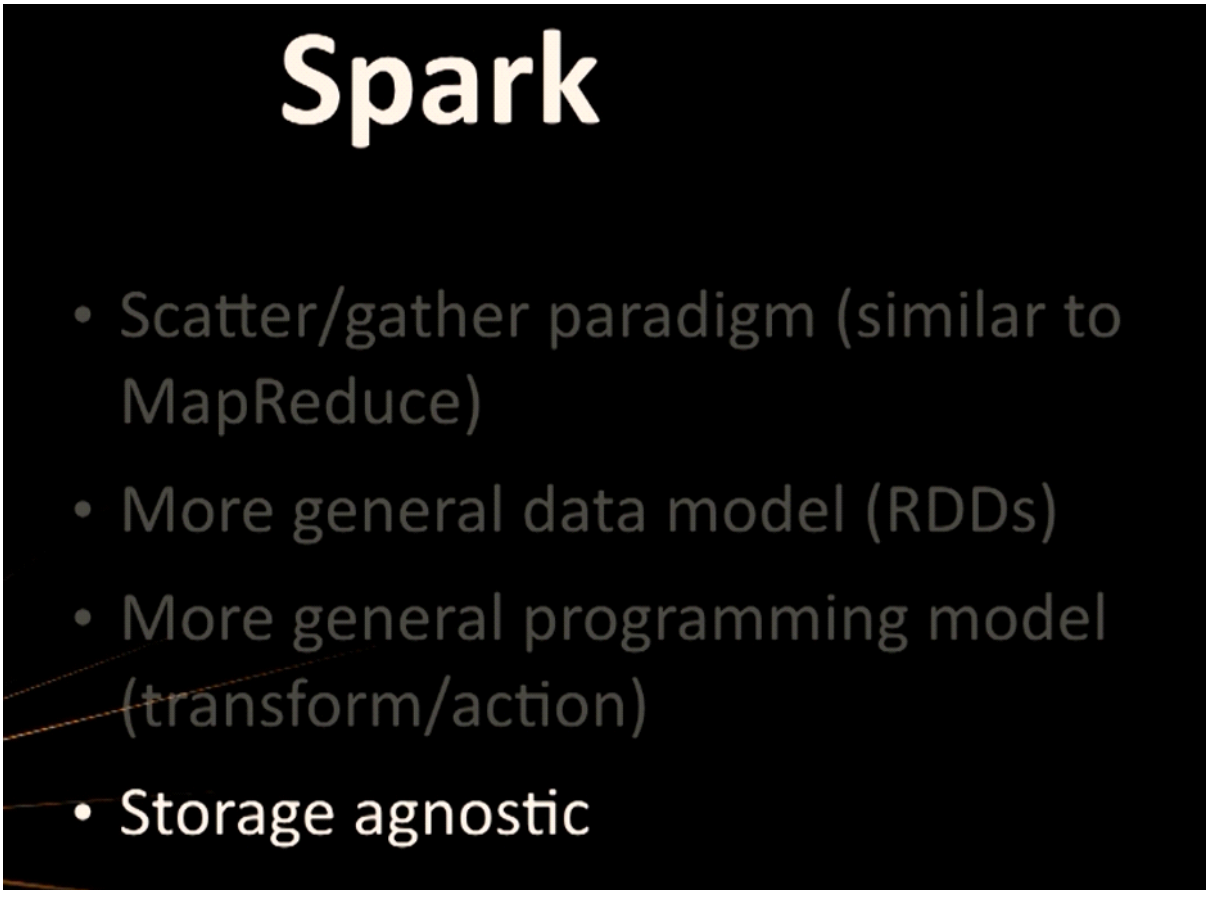

Sqoop
Pig
Hive
Impala
MapReduce
TEZ
Storm
Giraph
SolrCloud
Spark
YARN
HDFS

* APACHE SPARK STREAMING (scatter/gather)

3 февраля 2021 г. 19:39

https://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-12.pdf

- в отличие от MAP-REDUCE предоставляет огромное количество функций ТРАНСФОРМАЦИЙ
- реализует паттерн SCATTER-GATHER (те обобщение паттерна map-reduce)



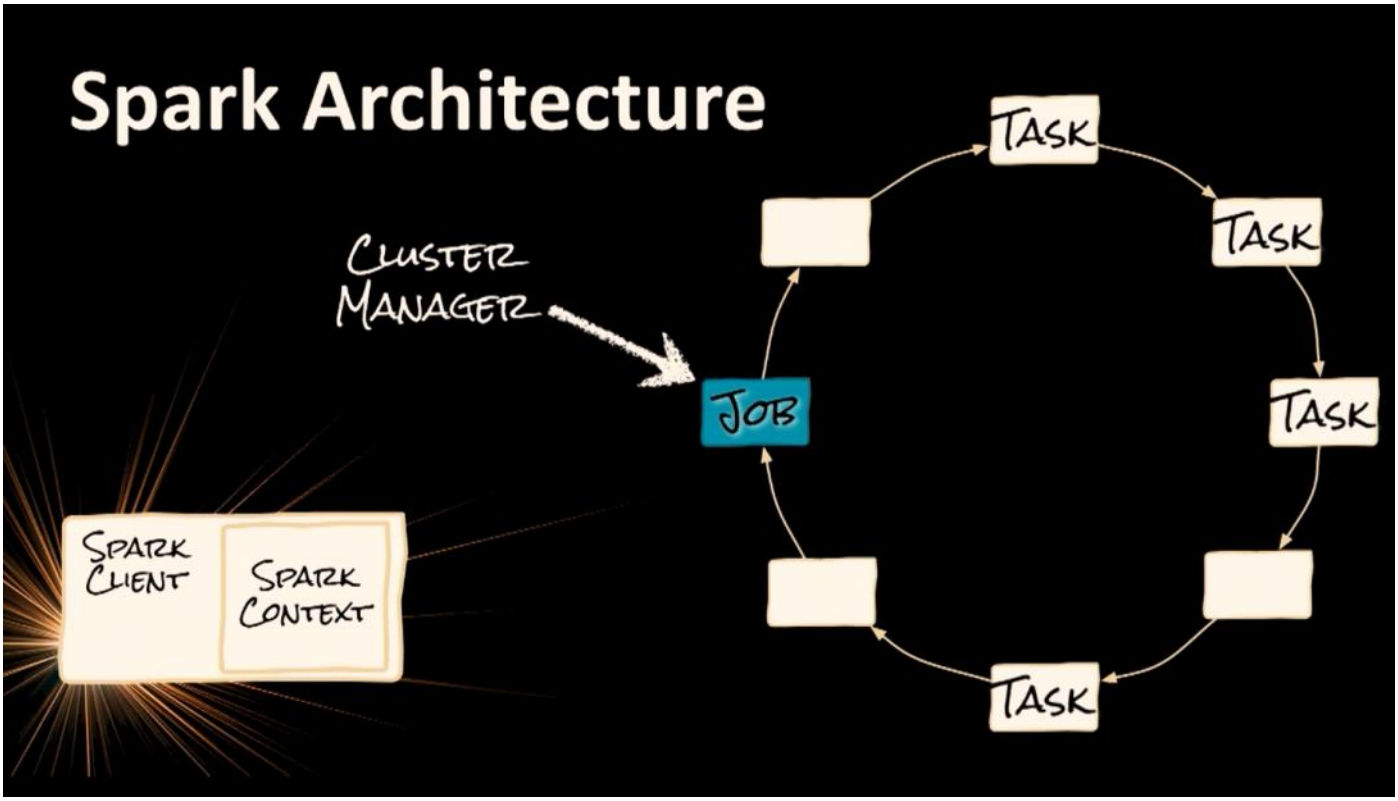
Spark Streaming - это расширение проекта Apache Spark для работы с непрерывными потоками событий. С технической точки зрения, Spark Streaming - это структура микропакетной обработки, а не структура потоковой обработки: Spark Streaming разделяет входящий поток событий на микропакеты, которые затем передаются в стандартный механизм Spark для обработки.

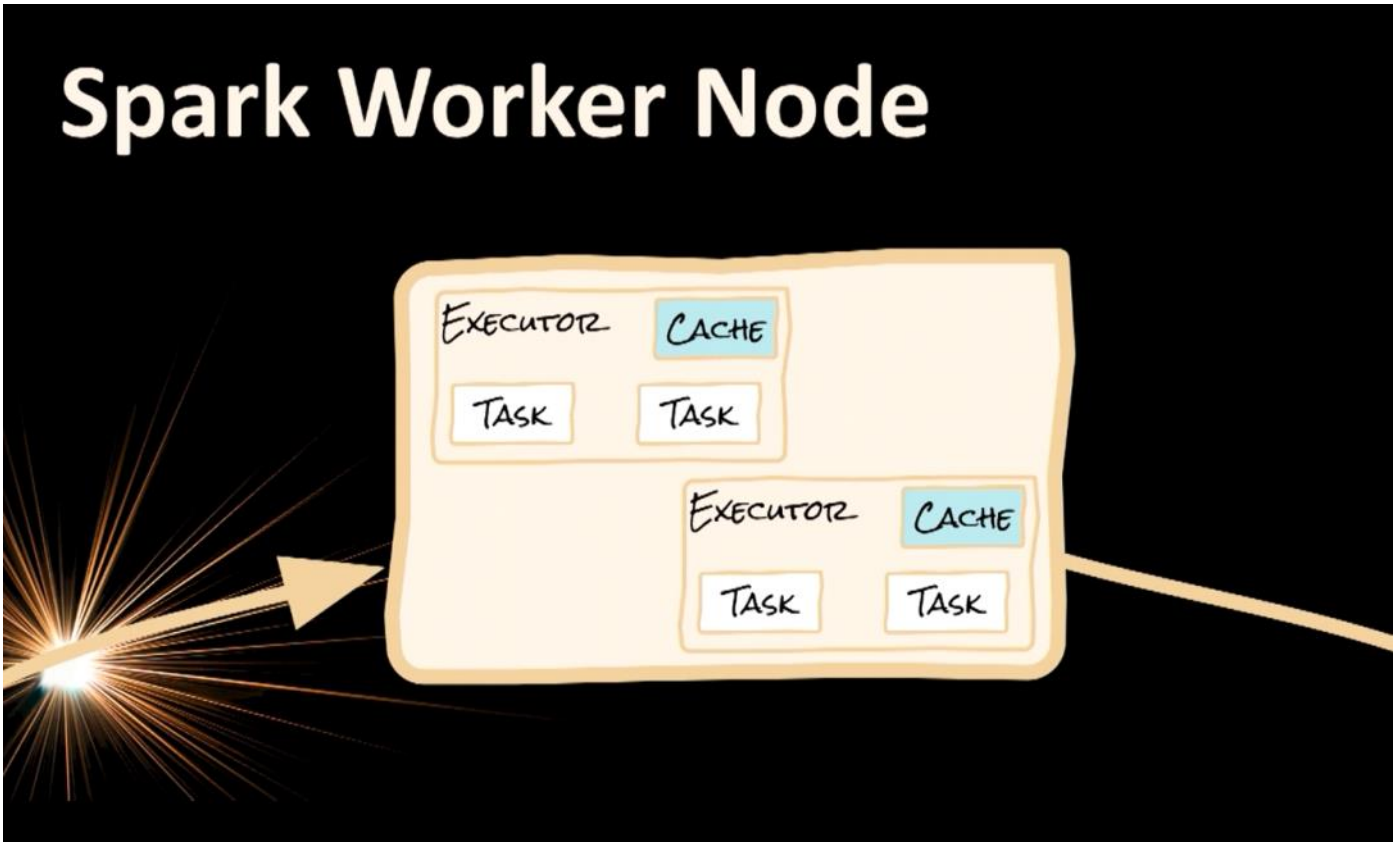
Микропакетирование в Spark Streaming дает нам возможность однократной обработки «бесплатно» и позволяет повторно использовать существующие возможности и код Spark, если они у нас есть. Но за это приходится платить: микропакетирование увеличивает задержку и снижает нашу гибкость в отношении управления окнами и масштабированием потока.

Spark Streaming не привязан к какой-либо одной потоковой технологии: почти любую технологию входящего потока можно нарезать и передать в Spark микропакетами; Spark Streaming "из коробки" поддерживает Flume, Kafka, Amazon Kinesis, файлы и сокеты, и вы можете написать свой собственный приемник, если хотите.

Для читателей, заинтересованных в более глубоком изучении Spark Streaming, Spark in Action Петара Зецевича и Марко Боначи и Streaming Data Эндрю Дж. Псалтиса, оба из которых опубликованы Manning, являются отличными ресурсами.

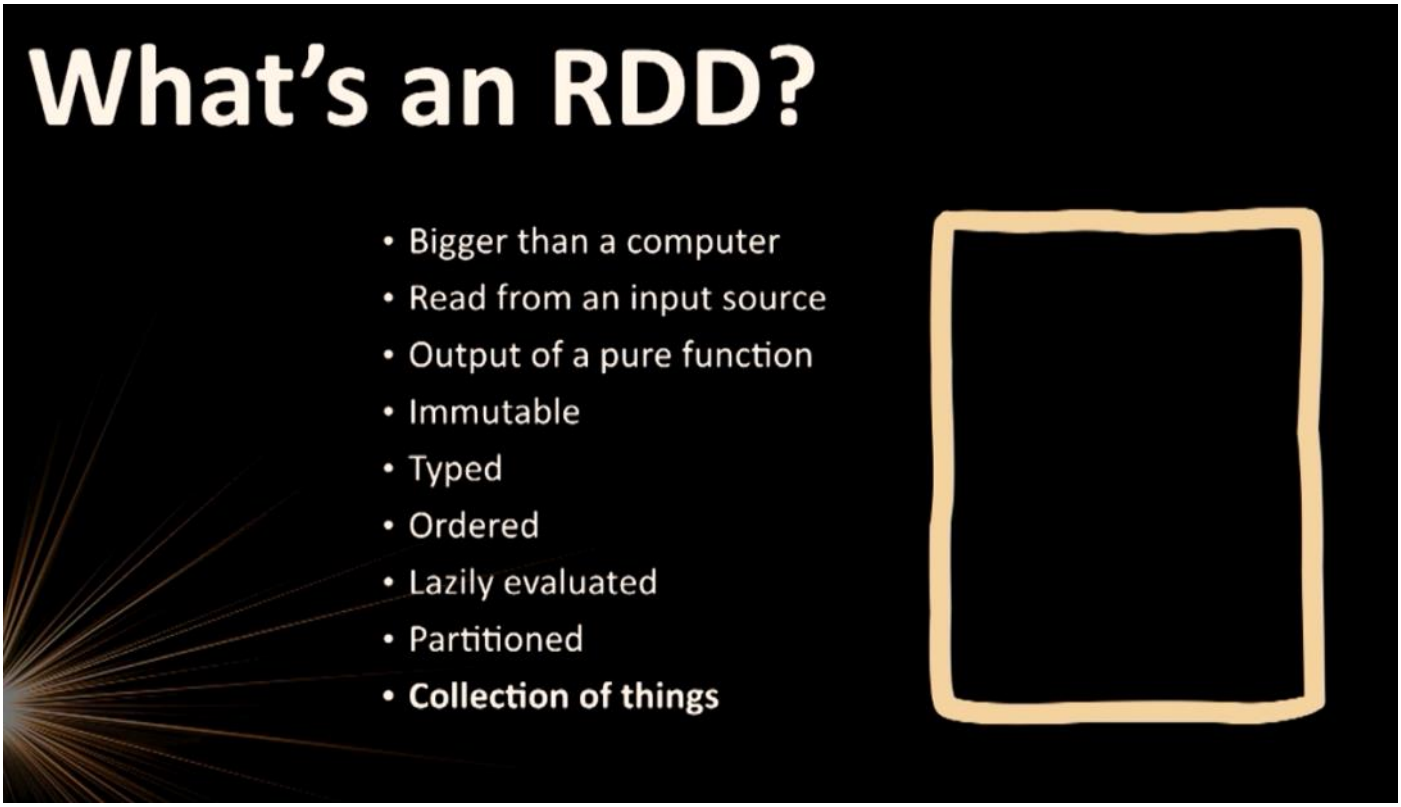
Используя повторяющиеся прогоны строго согласованного пакетного механизма для обеспечения непрерывной обработки неограниченных наборов данных, Spark Streaming доказал, что можно получить как правильность, так и результаты с малой задержкой, по крайней мере, для упорядоченных наборов данных.





storage agnostic - спарк может использовать любое хранилище данных (HDFS, Cassandra,)

- используя логическую абстракцию RDD
- функция трансформации преобразует один RDD в другой RDD



Первоначально известность Spark подпитывала его способность часто выполнять большую часть вычислений конвейера полностью в памяти, не касаясь диска

- Инженеры достигли этого с помощью идеи Resilient Distributed Dataset (RDD), которая в основном фиксировала всю происхождение данных в любой заданной точке конвейера, позволяя пересчитывать промежуточные результаты по мере необходимости при отказе машины при предположении, что а) ваши входные данные были всегда можно воспроизводить, и б) ваши вычисления были детерминированными.

spark тоже как преемник hadoop

- учитывая значительный прирост производительности, который пользователи смогли реализовать по сравнению со стандартными заданиями Hadoop. С этого момента Spark постепенно завоевал репутацию де-факто преемника Hadoop.

Spark Streaming

- Через несколько лет после создания Spark Татхагата Дас, тогда аспирант AMPLab, пришел к выводу, что: эй, у нас есть этот быстрый механизм пакетной обработки, а что, если мы просто подключим все, чтобы запустить несколько пакетов один за другим, и использовали это для обработки потоковых данных? Благодаря такому пониманию и родился Spark Streaming.

строгая консистентность позволяет отказаться от лямбда архитектуры

- Что было действительно фантастическим в Spark Streaming, так это то, что благодаря строго согласованному пакетному движку, обеспечивающему скрытые возможности, мир теперь имел движок потоковой обработки, который мог сам обеспечивать правильные результаты, не нуждаясь в помощи дополнительных пакетных заданий.
- Другими словами, при правильном варианте использования вы можете отказаться от своей системы Lambda Architecture и просто использовать Spark Streaming. Приветствую Spark Streaming!
- важным вкладом, который Spark внес в таблицу, стал тот факт, что это был первый общедоступный движок потоковой обработки с сильной семантикой согласованности , хотя и только в случае упорядоченных данных или времени события. - независимые вычисления.

Spark Streaming лучше всего подходит для упорядоченных данных или вычислений, не зависящих от времени события.

- И, как я неоднократно повторял в этой книге, эти условия не так распространены, как можно было бы надеяться при работе с широко распространенными сегодня крупномасштабными,

ориентированными на пользователя наборами данных.

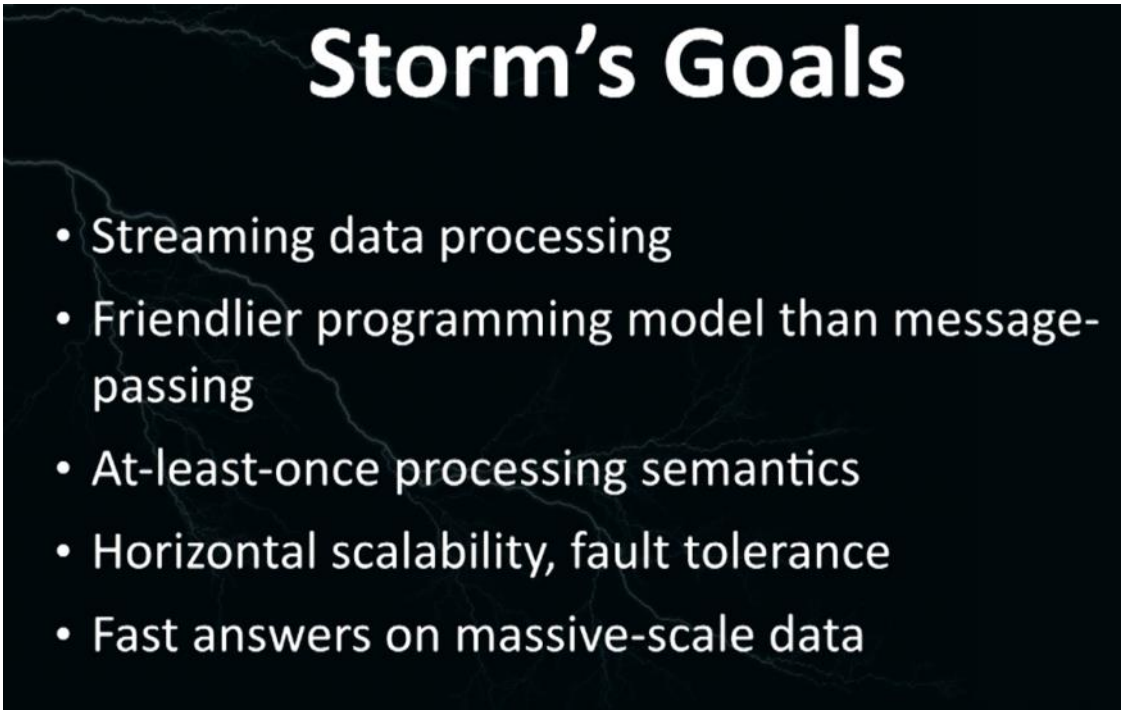
Spark Streaming основан на идее небольших повторяющихся запусков механизма пакетной обработки,

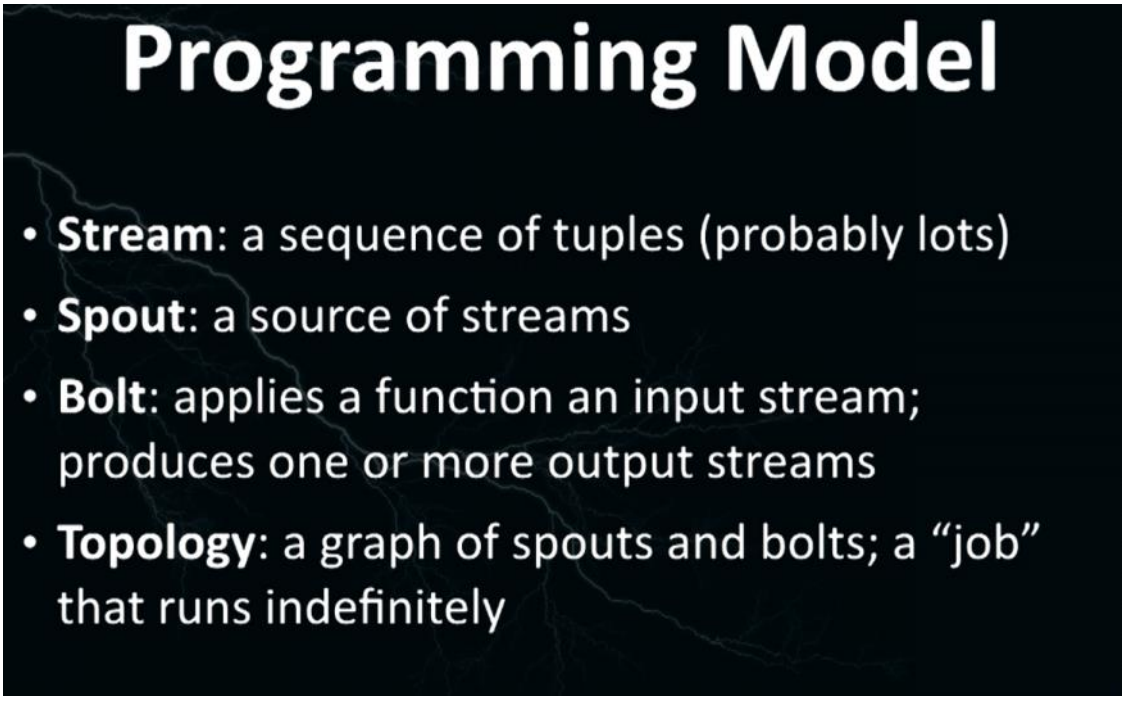
- недоброжелатели утверждают, что Spark Streaming не является истинным потоковым движком в том смысле, что прогресс в системе ограничивается глобальными барьерами каждой партии.

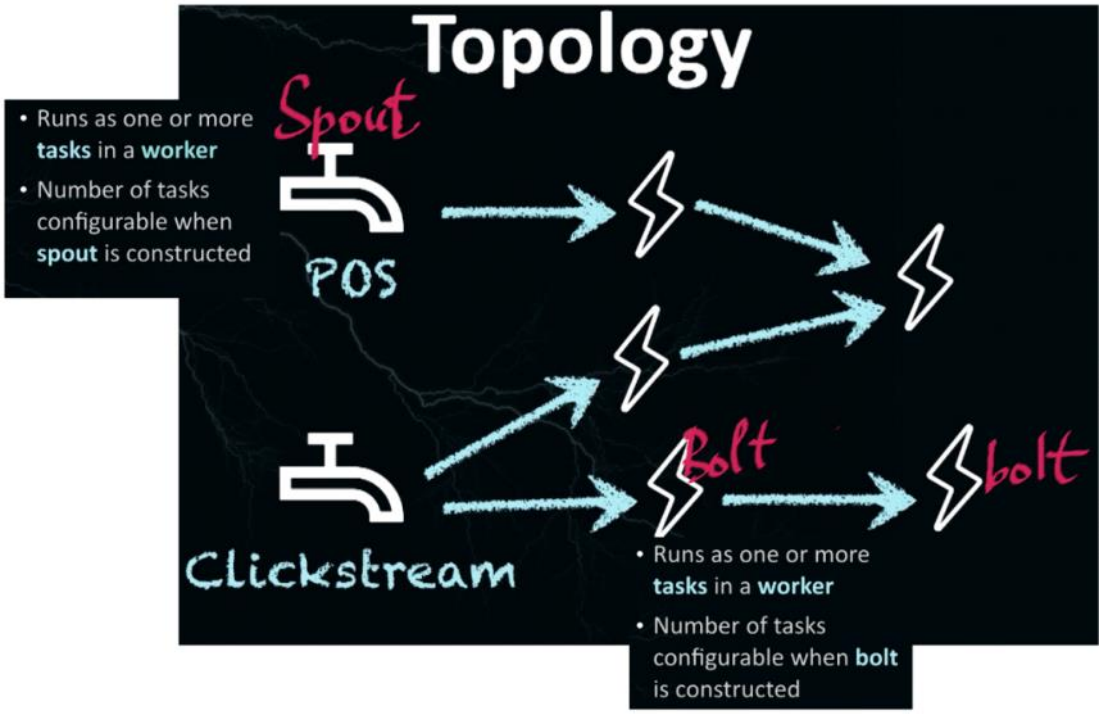
* APACHE STORM (lambda architecture)

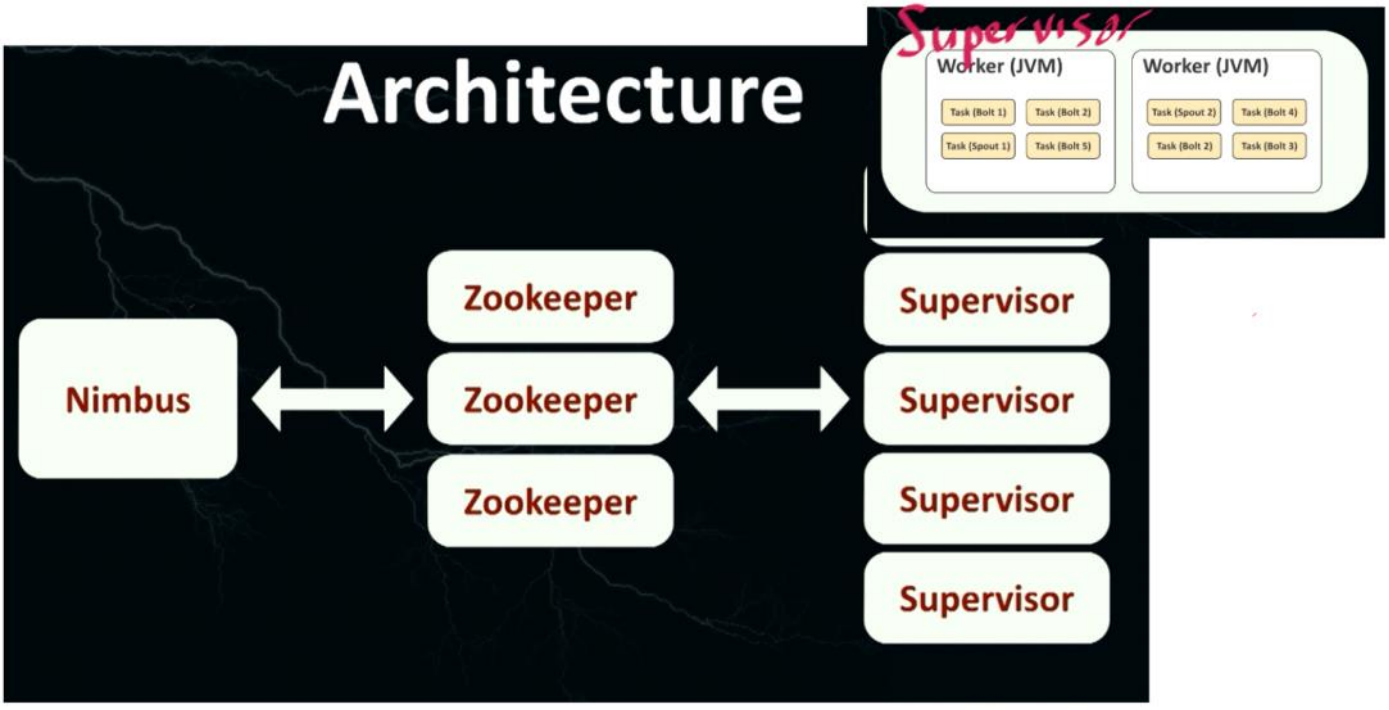
3 февраля 2021 г. 19:39

в отличие от пакетной обработки в SPARK он-STORM сделан специально для потоковой обработки









- **Nimbus**: central coordinator of jobs
- **Supervisor**: a node that performs processing
- **Task**: a thread of bolt or sprout execution
- **Worker**: a JVM process where a topology executes

первая потоковая система, которая получила действительно широкое распространение в отрасли,

http://nathanmarz.com/blog/history-of-apache-storm-and-lessons-learned.html

Apache Storm был первым представителем «новой волны» фреймворков потоковой обработки. Storm был написан Натаном Марцем из BackType; Twitter приобрел BackType и Storm с открытым исходным кодом в 2011 году. Будучи новаторской технологией, Storm действовал несколько иначе, чем его преемники:

Отказоустойчивость достигается за счет подтверждений записи с резервными копиями записей восходящего направления, а не за счет моментальных снимков состояния или контрольных точек.

Вместо того, чтобы использовать сторонний планировщик, Storm изначально создал свой собственный (Nimbus), хотя теперь Storm поддерживает как YARN, так и Mesos.

Storm представила отдельную библиотеку Storm Trident для поддержки однократной обработки.

Шторм был популяризирован книгой « Большие данные » Натана Марца и Джеймса Уоррена (Мэннинг) и продолжает широко использоваться; Система-преемница Twitter, Heron, и Apache Flink (рассмотренные далее в этом разделе) предлагают API-интерфейсы, совместимые со Storm, для облегчения принятия.

Принеся в жертву правильности результатов в пользу уменьшения задержки, Storm привнес в массы потоковую обработку, а также положил начало эре лямбда-архитектуры, в которой слабосогласованные механизмы потоковой обработки использовались вместе с строго согласованными пакетными системами для достижения истинной бизнес-цели низкой - задержка, в конечном итоге стабильные результаты.

idea of a "stream" as a distributed abstraction

- Streams would be produced and processed in parallel, but they could be represented in a single program as a single abstraction.

промежуточные брокеры

- ?Однако сама идея использования брокеров сообщений для промежуточных сообщений казалась неправильной, и я решил посидеть на Storm, пока не смогу лучше все обдумать.
    - У меня был инстинкт, что должен быть способ получить эту гарантию обработки сообщений без использования промежуточных брокеров сообщений.
- Причина, по которой я подумал, что мне нужны эти промежуточные брокеры, заключалась в предоставлении гарантий обработки данных. Если болт не смог обработать сообщение, он может воспроизвести его от любого брокера, от которого он получил сообщение.

Я долгое время пользовался Hadoop и использовал свои знания о дизайне Hadoop, чтобы улучшить дизайн Storm

- Например, одна из самых неприятных проблем, с которыми я столкнулся в Hadoop, заключалась в том, что в некоторых случаях рабочие Hadoop не закрывались, а процессы просто бездействовали. В конце концов, эти «зомби-процессы» будут накапливаться, поглощая ресурсы и делая кластер неработоспособным.
- Основная проблема заключалась в том, что Hadoop возлагал бремя остановки работника на самого работника, и по ряду причин работники иногда не могли выключить себя.
- еще одна проблема, с которой я столкнулся с Hadoop, заключалась в том, что если JobTracker по какой-либо причине прекращал работу, все выполняемые задания прекращались. Это был настоящий инструмент для удаления волос, когда у вас была работа, которая

выполнялась в течение многих дней. Со Storm было еще более неприемлемо иметь такую единую точку отказа, поскольку топологии предназначены для вечной работы. Поэтому я разработал Storm как «отказоустойчивый процесс»: если демон Storm будет убит и перезапущен, он абсолютно не повлияет на работающие топологии.

- В этом посте я небрежно назвал Storm «Hadoop реального времени», и эта фраза действительно прижилась. По сей день люди все еще используют его, и многие люди даже превращают его в «Hadoop реального времени». Этот случайный брендинг был действительно мощным и помог в дальнейшем внедрении.

Ключевой вопрос, с которым вы должны столкнуться, чтобы получить одобрение проекта, - это создание социальных доказательств.

- Социальное доказательство существует во многих формах: документированное использование проекта в реальном мире, наблюдатели Github, активность в списке рассылки, подписчики списков рассылки, подписчики в Twitter, сообщения в блогах о проекте и т. Д

Самая сложная часть создания проекта с открытым исходным кодом - это создание сообщества разработчиков, вносящих свой вклад в проект.

- Это определенно то, с чем я боролся
- К сожалению, «разработка, основанная на видении», имеет серьезный недостаток в том, что очень трудно создать активное и энергичное сообщество разработчиков. Во-первых, очень мало места для кого-либо, чтобы прийти и внести значительный вклад, поскольку я все контролирую. Во-вторых, я являюсь основным узким местом во всем развитии. Стало очень, очень трудно успевать за поступающими запросами на включение
- Еще одним недостатком сосредоточения разработки на себе было то, что люди рассматривали меня как единственную точку отказа для проекта. Люди высказывали мне опасения по поводу того, что произойдет, если меня сбит автобус.

- Я думаю, что «разработка, основанная на видении», лучше всего, когда пространство решений для проекта еще не полностью изучено. Так что для Storm было хорошо, что я контролировал все решения, пока мы создавали мультитенантность, настраиваемые метрики, Trident и основные рефакторинги производительности. Серьезные проблемы дизайна могут быть решены только тем, кто хорошо разбирается в проекте в целом.

Apache предоставит Storm мощный бренд,

- прочную юридическую основу и именно ту модель, основанную на консенсусе, которую я хотел для проекта.

обработка данных с малой задержкой и слабой согласованностью

weak consistency - ослабили консистентность

- The interesting thing about Storm, in comparison to the rest of the systems we've talked about so far, is that the team chose to loosen the strong consistency guarantees found in all of the other systems we've talked about so far as a way of providing lower latency.

так родилать лямбда-архитектура

Учитывая ограничения Storm, проницательные инженеры начали использовать слабо согласованный конвейер потоковой передачи Storm вместе с строго согласованным конвейером пакетной обработки Hadoop.

- Первый давал неточные результаты с малой задержкой,
- тогда как второй давал точные результаты с большой задержкой, оба из которых затем каким-то образом объединялись вместе, чтобы обеспечить единое и, в конечном итоге, согласованное представление результатов с низкой задержкой.

Storm была системой, которая впервые дала отрасли возможность попробовать обработку данных с малой задержкой,

- и влияние этого отражено в широком интересе к потоковым системам и их внедрении сегодня.

APACHE SAMZA (kafka)

3 февраля 2021 г. 19:39

Apache Samza - это фреймворк потоковой обработки с отслеживанием состояния, созданный командой LinkedIn, которая также создала Apache Kafka. Таким образом, Samza имеет тесную интеграцию с Apache Kafka, используя Kafka для резервного копирования состояния, хранящегося в базах данных RocksDB. Samza имеет относительно низкоуровневый API, позволяющий напрямую взаимодействовать с функциями оконного управления потоками и управления состоянием Samza. Samza способствует здание любой требуемый комплекс

топология потоковой обработки из множества более простых заданий Samza, чтение и обратная запись для отдельных тем Kafka.

Задания Samza изначально создавались для запуска из планировщика приложений YARN; в последнее время Samza также может быть встроена в виде библиотеки в обычное приложение, что позволяет вам выбирать собственный планировщик (или не планировать его). Samza также переросла свою изначальную взаимозависимость с Kafka и теперь поддерживает Amazon Kinesis.

Samza продолжает развиваться, это отличный обучающий инструмент для потоковой обработки, который используется крупными компаниями. Но потоки Кафки (о которых будет рассказано далее в этом разделе), возможно, украли часть грома Самзы.

Применяя концепцию долговечного журнала к проблеме потокового транспорта, Kafka вернул теплое, нечеткое ощущение возможности повторного воспроизведения, которое было потеряно из-за эфемерных потоковых транспортов, таких как RabbitMQ и TCP-сокеты. А популяризируя идеи теории потоков и таблиц, он помог пролить свет на концептуальные основы обработки данных в целом.

Apache Kafka's Kafka Streams systems limit themselves to what I refer to in Chapter 8 as "materialized view semantics,"

- essentially repeated updates to an eventually consistent output table.
- Materialized view semantics are great when you want to consume your output as a lookup table: any time you can just lookup a value in that table and be okay with the latest result as of query time, materialized views are a good fit.
- They are not, however, particularly well suited for use cases in which you want to consume your output as a bonafide stream. I refer to these as true streaming use cases, with anomaly detection being one of the better examples.
- В системах, в которых эти функции отсутствуют, это делается ради простоты, но за счет снижения возможностей.

в кафке вы можете вычислить идеальный водяной знак.

- Для более простых источников ввода, таких как статически разделенная тема Kafka, в которой каждый раздел записывается в возрастающем временном порядке (например, с помощью веб-интерфейсов, регистрирующих события в реальном времени), вы можете вычислить идеальный водяной знак.
- Для более сложных источников ввода, таких как динамический набор журналов ввода, эвристика может быть лучшим вариантом. Но в любом случае водяные знаки обеспечивают явное преимущество перед альтернативой использования времени обработки для рассуждения о полноте времени события, что, как показал опыт, служит примерно так же, как карта Лондона при попытке ориентироваться по улицам Каира.

кафка это it's not a data processing framework but instead a transport layer
- persistent streaming transport, implemented as a set of partitioned logs

- Kafka сыграл одну из самых влиятельных ролей в продвижении потоковой обработки из всех систем
- До Kafka в большинстве систем потоковой обработки для отправки данных использовались какие-то эфемерные системы очередей, такие как Rabbit MQ, или даже старые TCP-сокеты. Долговечность может быть в некоторой степени обеспечена за счет резервного копирования в восходящем направлении в производителях (т. Е. Возможность для вышестоящих производителей данных повторно отправлять данные, если последующие рабочие вышли из строя), но часто восходящие данные также хранились эфемерно. И большинство подходов полностью игнорировали идею возможности повторного воспроизведения входных данных позже
- Кафка все изменил. Взяв закаленную в боях концепцию долговечного журнала из мира баз данных и применив ее к области потоковой обработки, Kafka вернул нам всем чувство безопасности, которое мы потеряли при переходе от надежных источников ввода, распространенных в Hadoop / пакетная обработка эфемерных источников, преобладающих в то время в мире потоковой передачи.

replayability - Повторяемость - это основа, на которой построены сквозные гарантии exactly-once точного однократного выполнения в Apex, Flink, Kafka Streams, Spark и Storm.

- При выполнении в одноразовом режиме каждая из этих систем предполагает / требует, чтобы источник входных данных имел возможность перематывать и воспроизводить все

данные до самой последней контрольной точки. При использовании с источником ввода, который не обеспечивает такой возможности (даже если источник может гарантировать надежную доставку через резервное копирование восходящего потока), сквозная семантика «ровно один раз exactly-once» разваливается.

APACHE FLINK

3 февраля 2021 г. 20:22

Apache Flink - относительный новичок, но быстро становится серьезным конкурентом Spark и Spark Streaming. В отличие от Spark, который использует пакетную модель и обрабатывает варианты использования потоковой передачи с помощью микропакетов, Flink с самого начала создавался как потоковый движок: он обрабатывает пакетные данные, рассматривая их как ограниченный поток, имеющий начало и конец.

Как и Spark, Flink не привязан к Кафке. Flink поддерживает различные другие источники и приемники данных, включая Amazon Kinesis. Flink обладает развитыми возможностями управления потоками, поддерживает однократную обработку и богатый API запросов; Flink также внимательно следит за проектом Apache Beam, целью которого является предоставление стандартного полнофункционального API для взаимодействия с фреймворками потоковой обработки.

Быстро привнеся возможности обработки вне очереди в мир открытого исходного кода и объединив ее с собственными инновациями, такими как распределенные моментальные снимки и связанные с ними функции точек сохранения, Flink поднял планку потоковой обработки с открытым исходным кодом и помог возглавить текущие расходы. инноваций в области потоковой обработки в отрасли.

внедрение модели программирования Dataflow / Beam

высокоэффективная реализация моментальных снимков , которая дала ему надежные гарантии согласованности, необходимые для корректности.

Flink теперь имел возможность предоставлять семантику «точно один раз» вместе с встроенной поддержкой обработки event time (во время событий), был огромным в то время

- Но так было до тех пор, пока Джейми Гриер не опубликовал свою статью под названием «Расширение Yahoo! Streaming Benchmark » (рис. 10-30), что стало ясно, насколько производительным был Flink. В этой статье Джейми описал два впечатляющих достижения:

With the addition of a snapshotting mechanism, Flink gained the strong consistency needed for end-to-end exactly-once.

- Flink пошел еще дальше и использовал глобальный характер своих снимков, чтобы обеспечить возможность перезапуска всего конвейера из любой точки в прошлом, функция, известная как точки сохранения (описанная в разделе «Точки сохранения: возвращение времени вспять»). » пост Фабиан Hueske и Майкл Уинтерс [Рисунок 10-31]).
- Функция точек сохранения взяла теплую нечеткость длительного воспроизведения, которую Кафка применил к потоковому транспортному уровню, и расширила ее, чтобы охватить всю ширину конвейера.
-

flink быстрее чем storm

- Flink (с ровно один раз) достигает производительности в 7,5 раз больше, чем Storm (без ровно один раз). Кроме того, было показано, что производительность Flink ограничена из-за насыщения сети; Устранение узких мест в сети позволило Flink достичь почти в 40 раз большей пропускной способности, чем Storm.

the first practical streaming SQL API

Apache Beam

9 февраля 2021 г. 12:41

Предоставляя надежный уровень абстракции, который объединяет лучшие идеи отрасли, Beam обеспечивает уровень переносимости, позиционируемый как программный эквивалент декларативного лингва-франка, предоставляемого SQL, а также способствует принятию инновационных новых идей во всей отрасли.

Beam отличается от большинства других систем, описанных в этой главе, тем, что это в первую очередь модель программирования, API и уровень переносимости, а не полный стек с механизмом выполнения под ним.

- Основное видение Beam строится вокруг его ценности как уровня переносимости, и одной из наиболее привлекательных функций в этой области является запланированная поддержка полной межъязыковой переносимости. Хотя еще не полностью (но приземляется в скором времени), план Beam заключается в обеспечении достаточно производительной абстракции.

A unified batch plus streaming programming model, inherited from Cloud Dataflow where it originated

- Модель не зависит от каких-либо языковых реализаций или систем времени выполнения. Вы можете думать об этом как об эквиваленте Beam реляционной алгебре SQL.

Beam runners exist currently for Apex, Flink, Spark, and Google Cloud Dataflow