Arquitecturas y tecnologías para el *big data*

Marta Zorrilla – Diego García-Saiz

Enero 2017

Tabla de contenidos

- Evolución histórica
- El ciclo del *Big Data*
- Arquitecturas
  - Lambda
  - Kappa
- Tecnologías

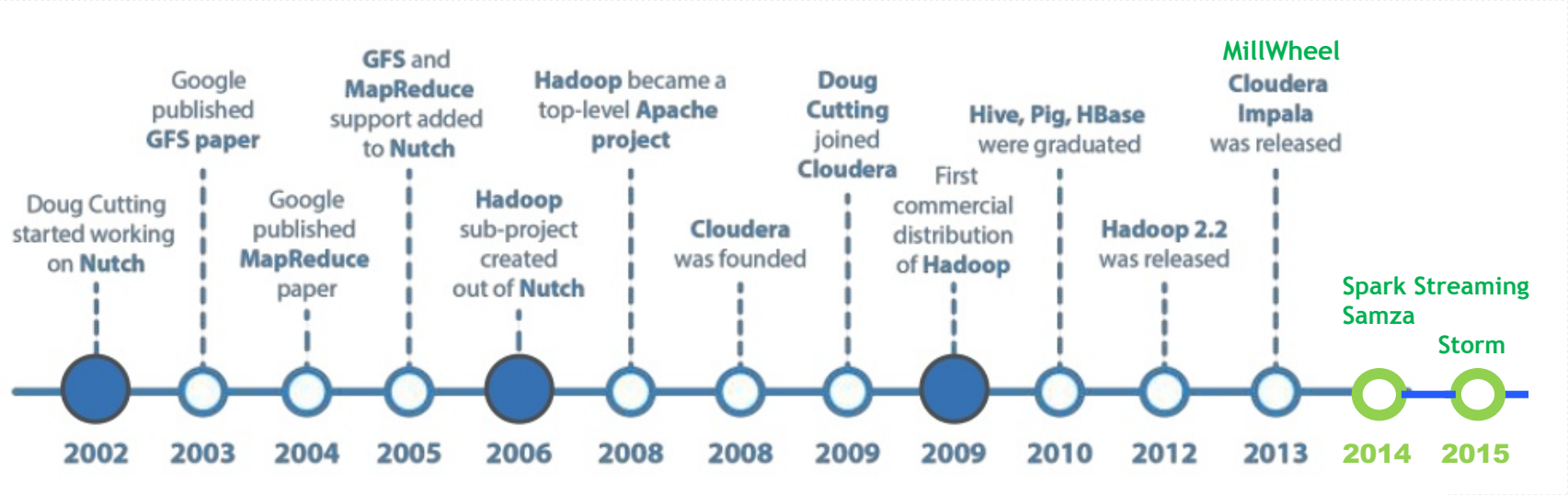
Bibliografía

- Libros
  - T. Dunning & E. Friedman: “Streaming Architecture”. O’Reilly, 2016
  - Harrison et al. Next Generation Databases: NoSQL, NewSQL, and Big Data. 2015. Apress.
  - Nathan Marz, James Warren. Big Data: Principles and best practices of scalable realtime data systems 1st Edition. 2015. Manning publisher
  - N. Narkhede, G Shaohira & T. Palino: “Kafka: The definitive Guide”. O’Reilly, 2017 (Early release)
  - H.Karau, A. Konwisnski, P. Wendell & M. Zaharia: “Learning Spark”. O’Reilly, 2015.

Bibliografía

- Artículos
  - R. Cattell. Scalable SQL and NoSQL Data Stores. SIGMOD Record 39(4), 2010
  - Chen et al (2016) Realtime data processing at Facebook, SIGMOD '16 Pages 1087-1098
  - Wolfram Wingerath, Felix Gessert, Steffen Friedrich, and Norbert Ritter. Real-time stream processing for Big Data. Information Technology 2016; 58(4): 186–194
  - Giacinto Caldarola and Rinaldi. Big Data: A Survey - The New Paradigms, Methodologies and Tools. In Proc. of 4th International Conference on Data Management Technologies and Applications - Volume 1: KomIS, 362-370, 2015, Colmar, Alsace, France

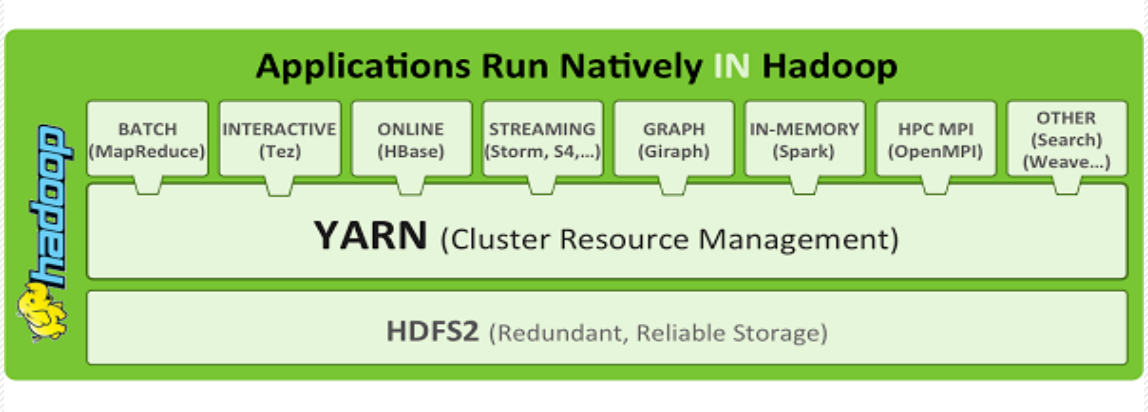
Google: pionero del *Big Data*



Fuente: Harrison et al (2015)

Hadoop ecosystem 2.0

Open-source framework for distributed computing



Fuente: http://www.kdnuggets.com

Distribuciones:

- ✓ Cloudera
- ✓ Hortonworks
- ✓ MapR

Hadoop ecosystem 2.0

- **Hadoop Common-** conjunto de bibliotecas comunes y utilidades utilizadas por otros módulos de Hadoop.
- **HDFS-** La capa de almacenamiento predeterminada para Hadoop.
- **MapReduce-** Ejecuta una amplia gama de funciones analíticas analizando los conjuntos de datos en paralelo antes de "reducir" los resultados. La operación "Map" distribuye la consulta a diferentes nodos y la opción “Reduce" reúne los resultados para ofrecer un solo valor. Tecnología Batch
- **YARN-** Se encarga de la gestión de recursos del clúster.
- **HBase-** BD columnar diseñada para ejecutarse en HDFS.

http://hadoop.apache.org/

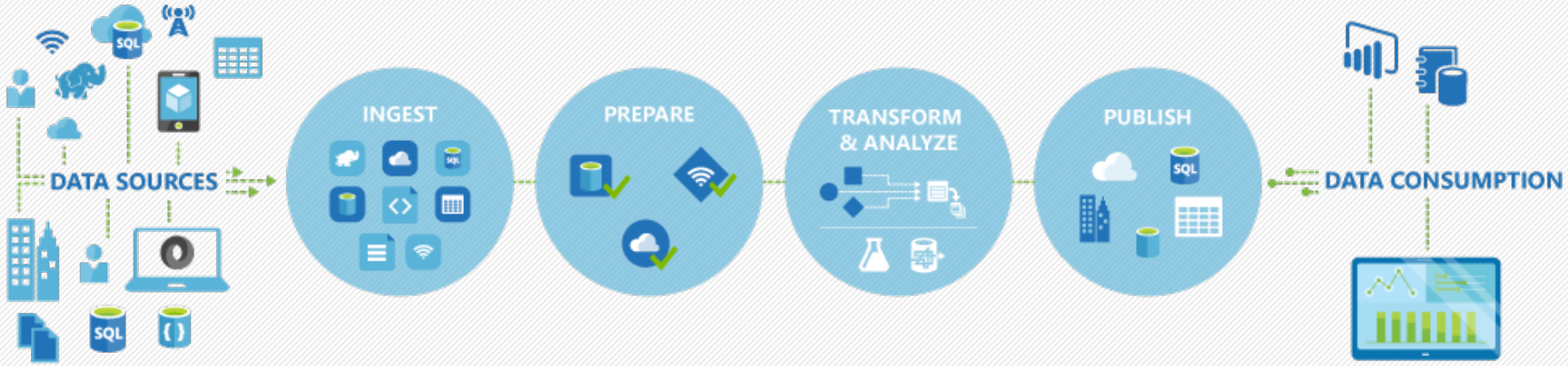
Hadoop ecosystem 2.0

- **Spark-** procesado de datos a gran escala. Trabaja en memoria. Particularmente apto para algoritmos de aprendizaje máquina. Soporta SQL. Funciona interactivamente con Scala, Python y R.
- **Mahout-** es una biblioteca de aprendizaje automático. Trabaja sobre MapReduce y Spark.
- **Hive-** es una infraestructura de data warehouse construida sobre Hadoop. Proporciona un lenguaje simple, HiveQL, manteniendo el soporte completo de MapReduce. Facilita que los programadores SQL con poca experiencia previa con Hadoop pueden utilizar el sistema más fácilmente.
- **Flume-** es un servicio distribuido, confiable y disponible para recopilar, agregar y mover eficientemente grandes cantidades de datos de log. Tiene una arquitectura simple y flexible basada en flujos de datos en streaming.

Hadoop ecosystem 2.0

- **Pig** es un lenguaje para desarrollar aplicaciones en el entorno Hadoop. Pig es una alternativa a la programación Java para MapReduce, y genera automáticamente funciones MapReduce
- **Sqoop** es una herramienta que ayuda en la transición de datos de otros sistemas de bases de datos (p.ej. relacionales) a Hadoop.
- **Oozie** es el planificador de flujos de trabajo.
- **Impala** - base de datos analítica nativa para Apache Hadoop.
- **Kafka** – publicador/subscriptor de mensajes.
- **Tez** - framework de programación de flujo de datos sobre Hadoop YARN
- **Etc…..**

El ciclo del Big data

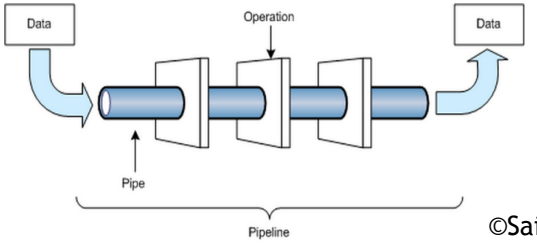


© Azure data factory

Workflows or Pipelines

- Secuencia de tareas que debe ser realizadas por un ordenador.
- Utilizados desde hace muchos años para la ingesta, transformación y carga de datos en otros sistemas (ETLs) o bien explotación directa por aplicaciones de usuario.
- Las diferencias entre el workflow tradicional y el *big data* pipeline son:
  - La tecnología *big data* permite escalar de forma que todos los datos de la organización puedan ser almacenados en un solo repositorio (*Data Lake*).
  - Se trabaja con tecnologías no relacionales (NoSQL), la cuales no require un esquema predefinido, sino que se crea en función de las necesidades del momento.
  - Tecnologías orientadas al procesamiento en tiempo real ("*in-memory*") y sobre flujos continuos de datos ("*streaming*").
  - El número de herramientas disponibles es muy elevado y el framework de trabajo admite diferentes configuraciones.

Big data pipeline

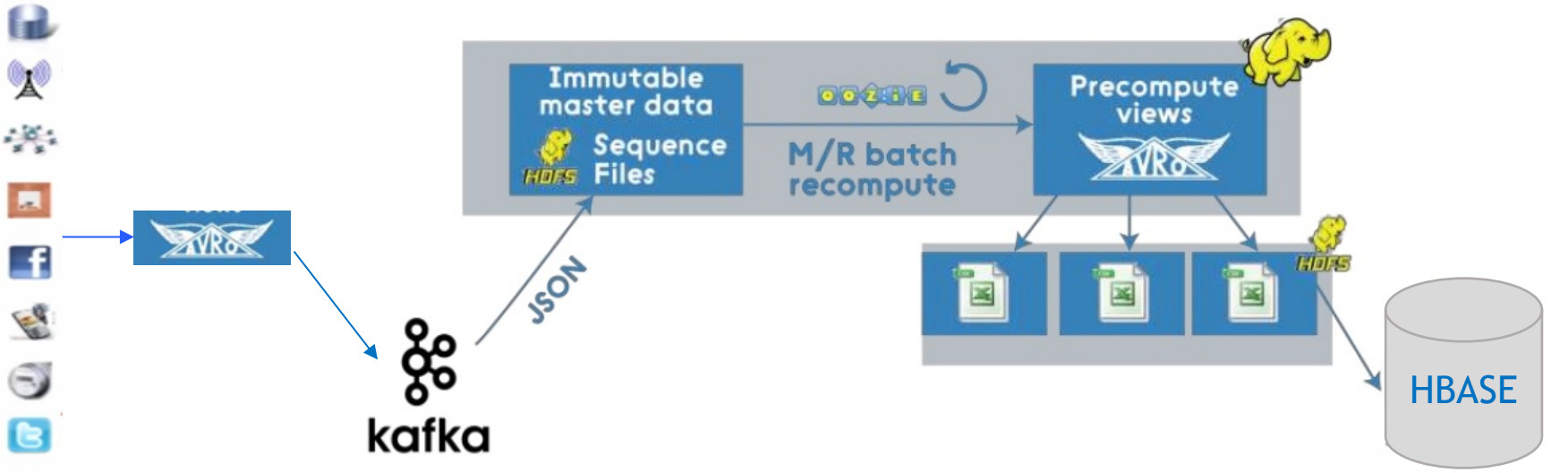


©Saini

M.Zorrilla - D.García

- Sistema que captura **eventos** para su análisis posterior
- Eventos:
  - información que proviene de logs de aplicaciones (clics en pág. Web, acción del usuario, envío de mensajes,…)
- Componentes:
  - Generador y serializador de eventos (p.ej. AVRO)
  - Bus de mensajes (p.ej. Kaftka)
  - Capa de procesamiento (p.ej. Spark)
  - Coordinador de las tareas en el workflow (p.ej. Oozie)
  - Persistencia (p.ej. MemSQL, Cassandra, HDFS,..)

Hadoop pipeline: versión inicial



Esta arquitectura no es válida para todos los tipos de trabajo:

- No procesa transacciones (acceso aleatorio)
- Solo válido si el trabajo puede ser paralelizado
- No adecuado para acceso a datos con requisitos de baja latencia
- No adecuado para procesar muchos ficheros pequeños
- No adecuado para cálculos computacionales exigentes con pocos datos
- Trabaja con ficheros → no para “real time”

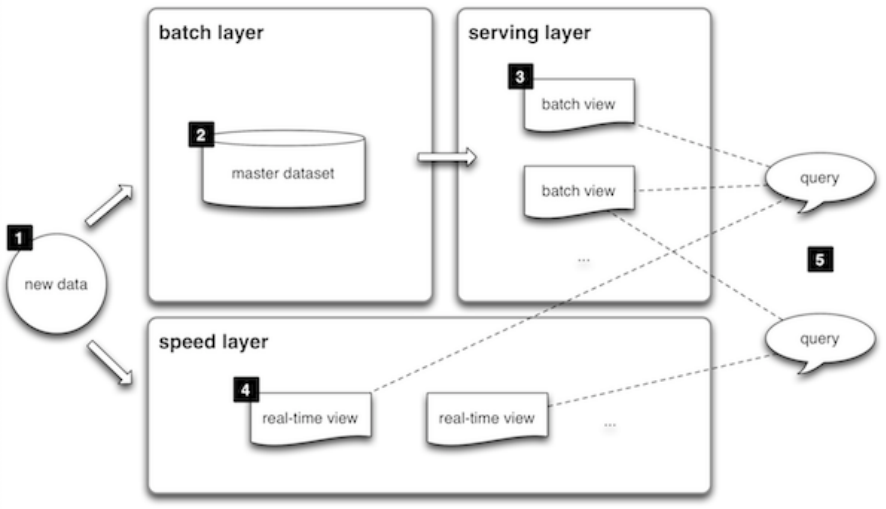
Pero hoy se necesita “REAL TIME”

I need fast access to historical data on the fly for predictive modeling with real time data from the stream

Arquitectura Lambda (λ)

Fuente: http://lambda-architecture.net/

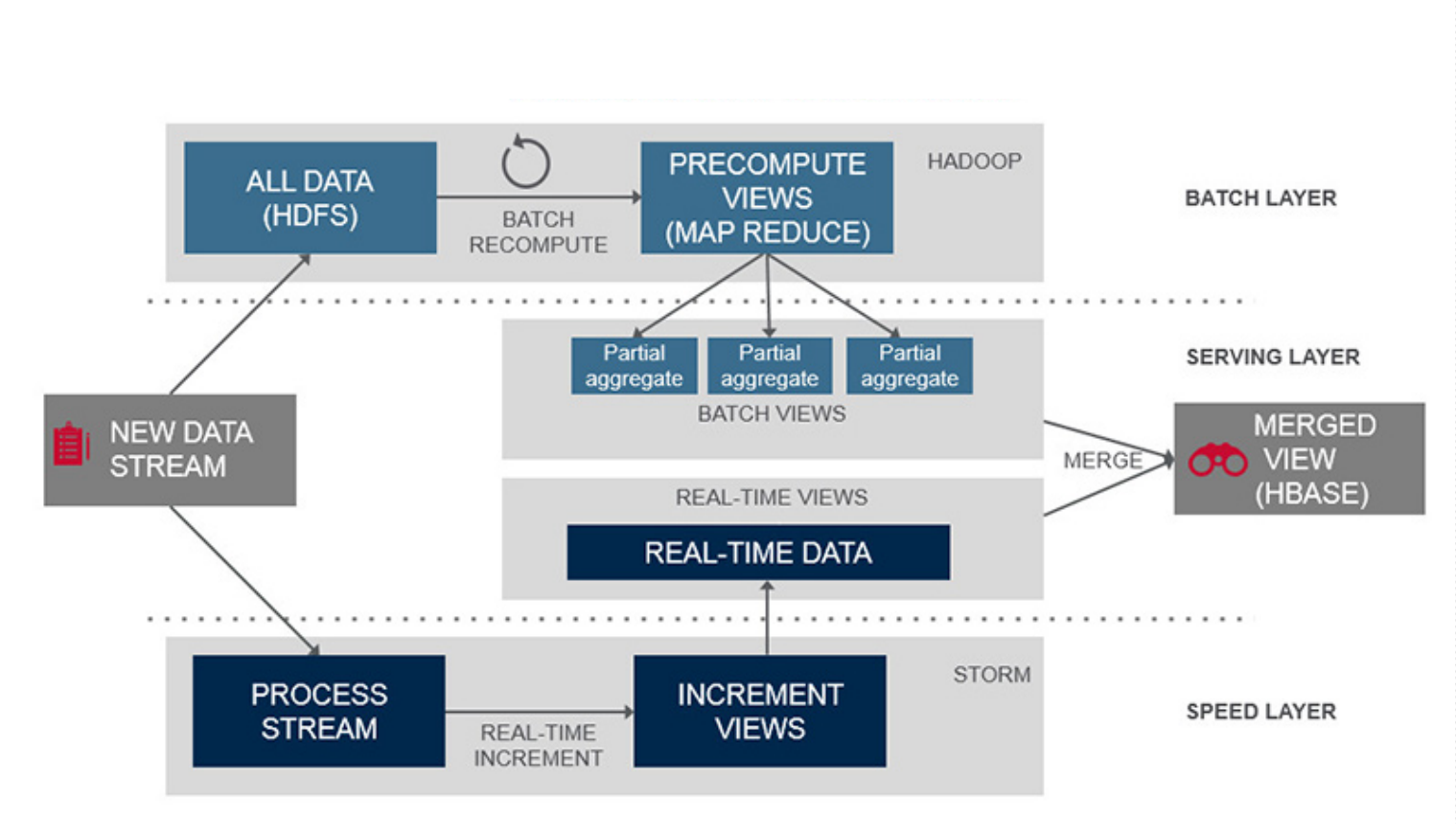
Arquitectura de procesamiento de datos distribuido, escalable y tolerante a fallos que combina procesado en *batch* y en *streaming* utilizando "*commodity hardware*"



- Los datos se envían tanto a la capa de ***batch*** como a la capa ***speed*** para su procesamiento.
- La capa ***batch*** tiene dos funciones: (i) gestionar el conjunto de datos maestro (*an immutable, append-only set of raw data*), y (ii) precomputar las vistas *batch*.
- La capa ***serving*** indexa las vistas *batch* para que puedan consultarse en modo de baja latencia y de manera ad hoc.
- La capa ***speed*** se encarga de las peticiones sujetas a baja latencia. Trabaja sobre datos recientes aplicando algoritmos rápidos e incrementales.
- Cualquier consulta se puede responder combinando los resultados de las vistas en *batch* y las vistas en tiempo real.

Arquitectura Lambda (λ)

Diferentes configuraciones y alternativas tecnológicas



Fuente: https://www.mapr.com/developercentral/lambda-architecture

Arquitectura Lambda (λ)

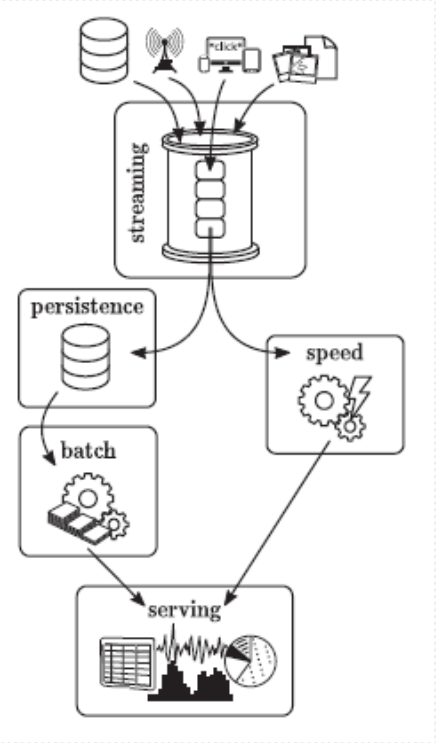
- Ventajas:
  - Conserva los datos de entrada sin cambios.
  - Esto permite reprocesar los datos cuando se producen cambios de criterio.

- Desventajas:
  - Mantener código diferente para producir el mismo resultado de dos sistemas distribuidos complejos (*batch* y *speed*) es costoso
    - Código muy diferente para MapReduce y Storm/Apache Spark
  - Además, no sólo se trata de código diferente, sino también de depuración e interacción con otros productos.
  - Al final es un problema sobre paradigmas de programación divergentes y diferentes.

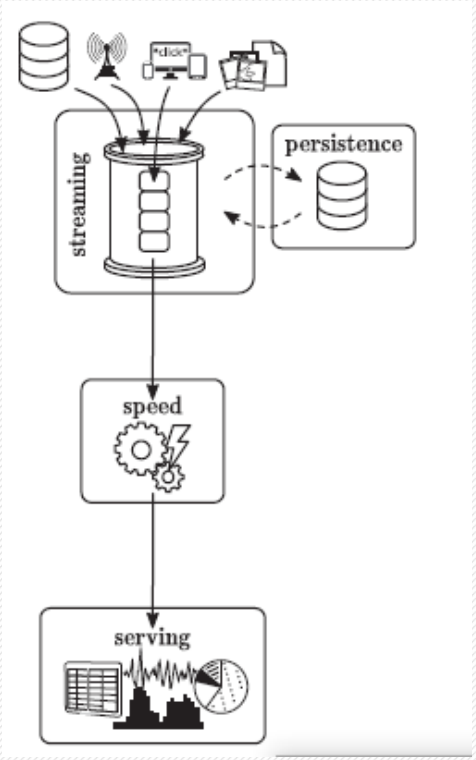
Arquitectura Kappa

- Jay Kreps acuñó este término en 2014

Lambda



Kappa



(©Wingerath et al, 2016)

Arquitectura Kappa

- Estrategia de funcionamiento:
  - Un sistema de mensajería (p.e. kafka) que mantenga el log de datos a procesar.
  - Si se necesita reprocesar, se comienza el trabajo en el instante de datos requerido y su resultado se vuelca en otra tabla.
  - Cuando este reprocesamiento finaliza, la aplicación comienza a leer de esta nueva tabla y se elimina la anterior.
- Ventajas:
  - Solo se recomputa cuando hay un cambio en el código.
  - Solo se mantiene un código.
  - Se puede volcar los datos de kafka a HDFS (disco), si hay limitaciones de memoria.

Arquitectura Kappa

M.Zorrilla - D.García

Configuraciones diversas

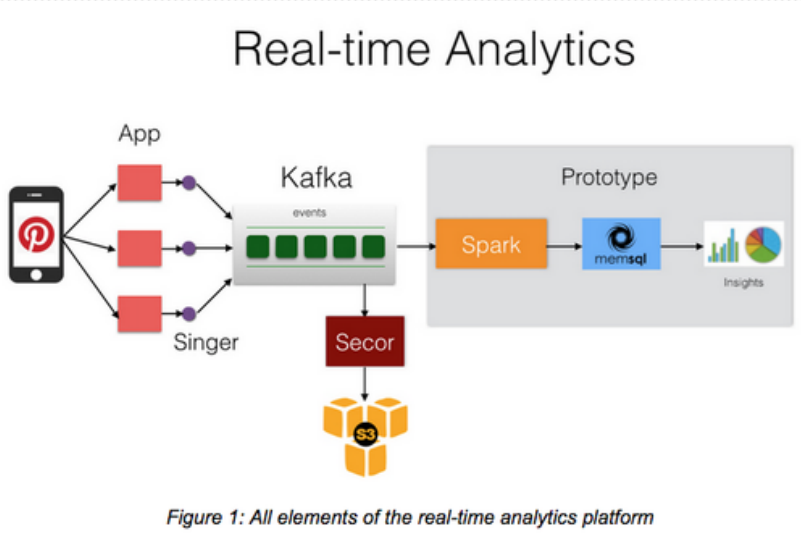


Figure 1: All elements of the real-time analytics platform

Fuente: https://medium.com/@Pinterest_Engineering/real-time-analytics-at-pinterest-1ef11fdb1099#.qq1ub9uhx

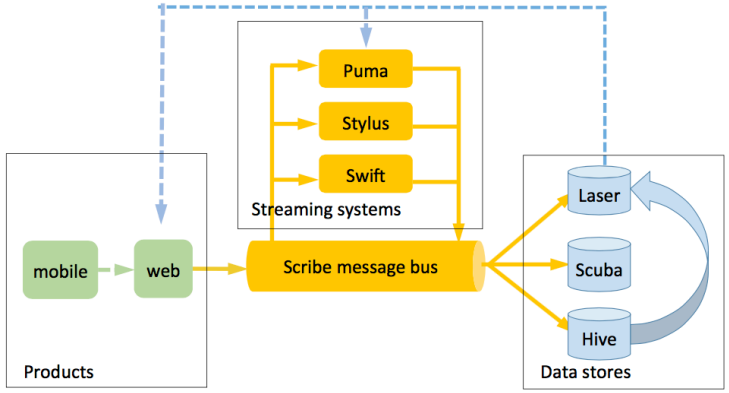


Figure 1: An overview of the systems involved in realtime data processing: from logging in mobile and web products on the left, through Scribe and realtime stream processors in the middle, to data stores for analysis on the right.

Fuente: Chen et al (2016) Realtime data processing at Facebook

Lambda vs Kappa

- Lambda:
  - Orientada a la analítica de datos tradicional.
  - Periódicamente (p.ej. diariamente) se recogen y procesan grandes volúmenes de datos estáticos (*batch*) al tiempo que se procesan "*on the fly*" datos dinámicos (*streaming*), combinando así volumen y velocidad.
- Kappa:
  - Orientada a la analítica en tiempo real (*soft strict*).
  - Se evita almacenar los datos y se procesan en cuanto se reciben minimizando el tiempo que el dato está en el pipeline.
  - La idea aquí es no recomputar todos los datos en la capa *batch*, sino hacerlo en la capa *streaming* y únicamente recomputar si se produce un cambio en la lógica de negocio.
- Ambas se construyen, generalmente, con componentes distribuidos desarrollados sobre la JVM (Java Virtual Machine), por tanto, son *soft-real-time systems* (orden de segundos).

Stream management

- “Class of software systems that deals with processing streams of high volume messages with very low latency.”

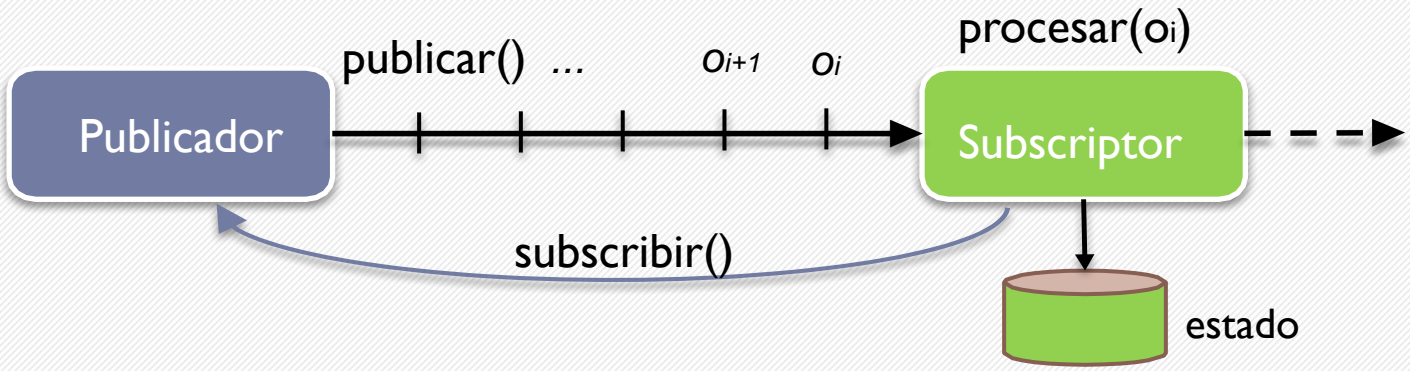
  Michael Stonebraker, Encyclopedia

Características de datos en streaming

- La tasa de llegada no está bajo el control del sistema, en general, es más rápido que el tiempo de procesamiento
- Los algoritmos deben trabajar con una sola pasada de los datos
- Requisitos de memoria sin límites → Se necesita una reducción drástica
- Mantener los datos en movimiento → Sólo almacenamiento volátil
- Soporte para aplicaciones en tiempo real
  - La latencia de 1 segundo es inaceptable → Necesidad de escalar y paralelizar
- Orden de llegada no garantizada → Algunos datos pueden retrasarse
- Deben asumirse imperfecciones en los datos
- Los datos (características) evolucionan con el tiempo
- Respuestas aproximadas (no exactas) son aceptables → Los resultados deben ser predecibles

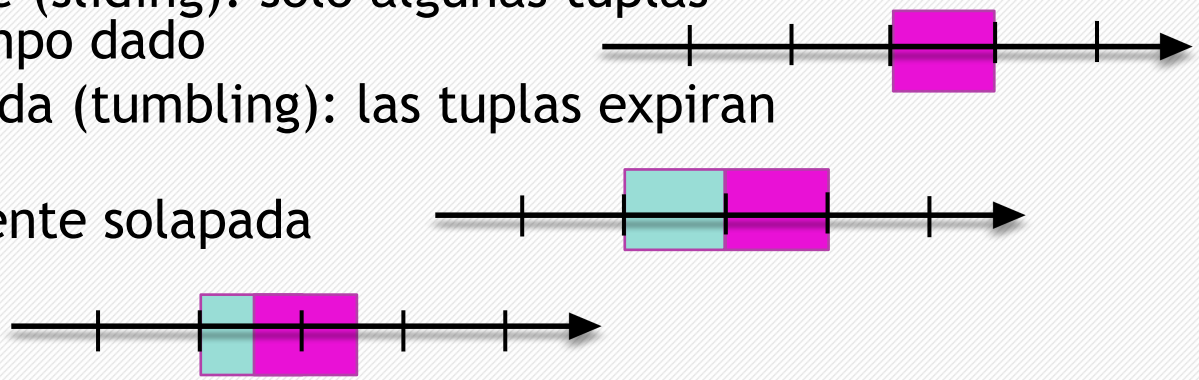
Data stream: principios básicos

- Data stream := < o_i, o_{i+1}, o_{i+2}, ... >
- Elemento de stream : o_i = (*data items*, *timestamp*)
- Paradigma publicar-subscribir



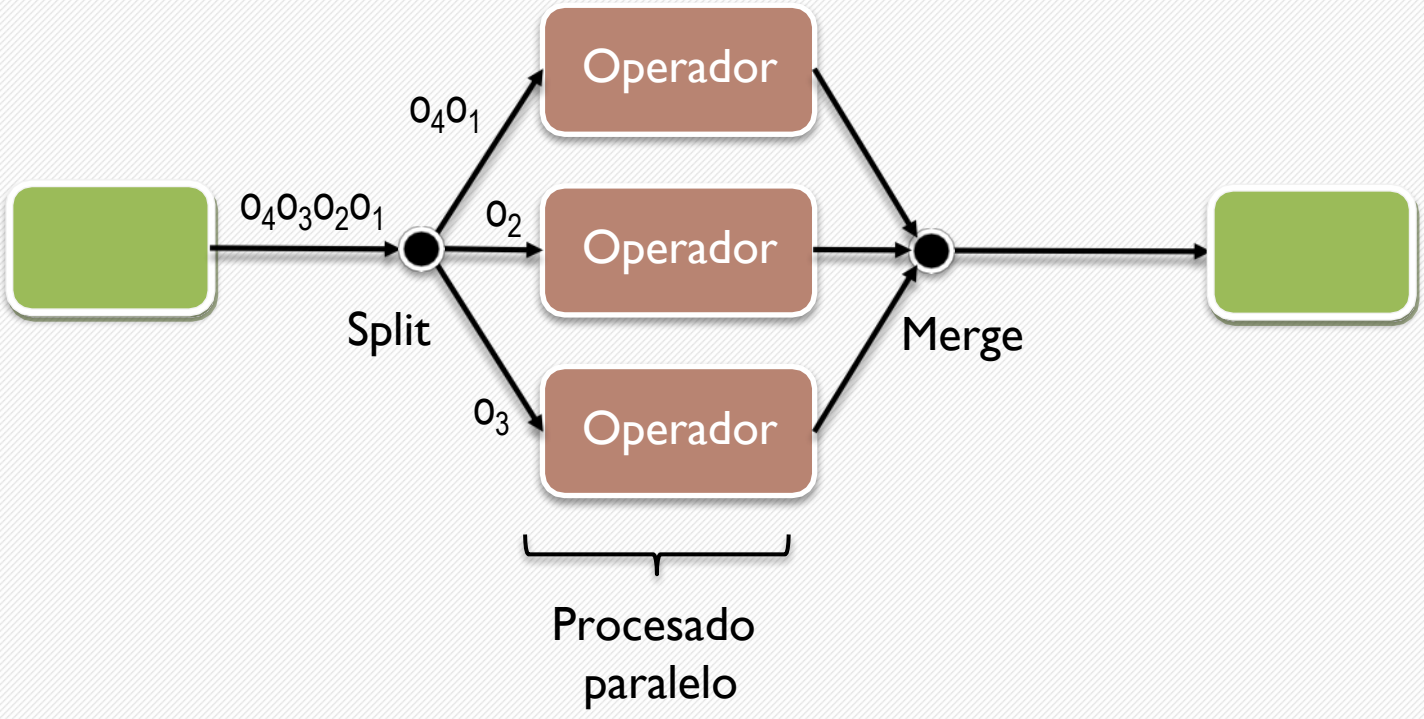
Ventanas en data streams

- Análisis de datos por ventanas: agregaciones, joins, etc
- Diferentes semánticas posibles:
  - Time-based: last 5 minutes
  - Count-based: 100 data items
  - Data-driven: Web session
- Estrategias:
  - Ventana deslizante (sliding): solo algunas tuplas expiran en un tiempo dado
  - Ventana no solapada (tumbling): las tuplas expiran todas al tiempo
  - Ventana parcialmente solapada



Paralelizado

- Gestión no trivial:
  - Particionado
  - Ordenación
  - Gestión del estado
  - Elasticidad



Tolerancia a fallos

- Pérdida de mensajes
- Garantía de entrega de mensajes (ACK)
  - At-least-once
  - At-most-once
  - Exactly-once
- Nodos fallan → procesado redundante y recuperación
- Diferentes estrategias:
  - Checkpoint/log
  - Buffers y retransmisión

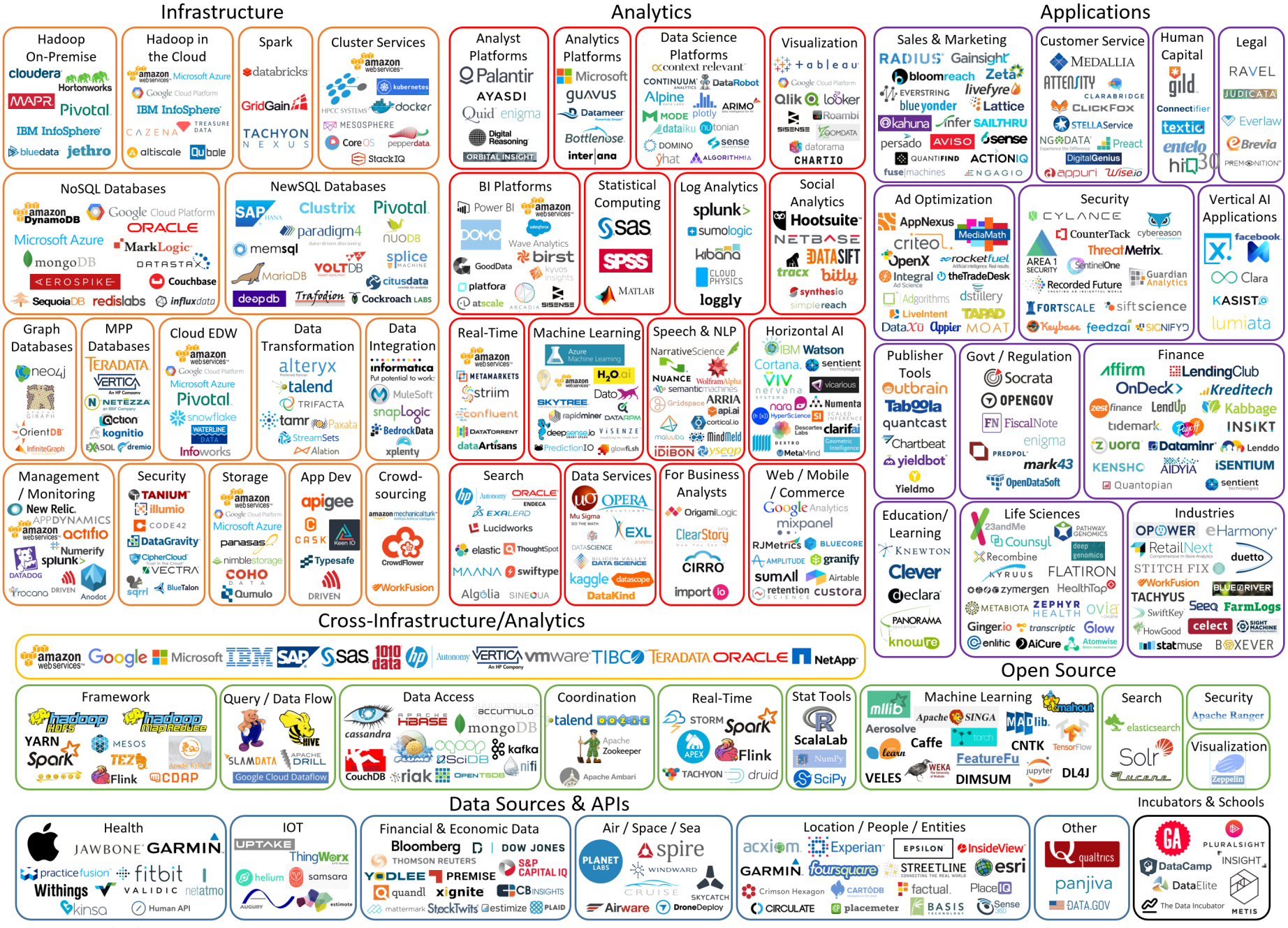
Data streaming processors: comparación

Nombre	Modelo computacional	Gestión del estado	Tolerancia a fallos	Orden de eventos
Storm -Twitter	Tuplas	Stateless	At-least-once, ACK por tupla	No
Storm+Trident	Micro-batch	Stateful	Exactly-once, ACK por tupla	Entre batches
Heron (sobre Storm)- 2015	Tuplas	Stateless	At-most-once, at-least-once and going to exactly-once, ACK por tupla	No
Flink	Tuplas	Stateful	Exactly-once semantics using checkpoints	No
Spark Streaming	Micro-batch	Stateful	Exactly-once semantics using checkpoints	Entre batches
Samza (LinkedIn)	Tuplas	Stateful	At-least-once	Dentro de la partición
MillWheel (Google)	Tuplas	Stateful	Exactly-once	Si

En resumen

- El problema resuelto por las arquitecturas *big data*, para algunos casos de uso, es el mismo que las plataformas de almacenes de datos tradicionales, pero las soluciones son órdenes de magnitud más potentes y eficientes.
- El procesamiento en tiempo real, como su nombre indica, implica realizar cálculos o procesar datos que llegan al sistema en tiempo real.
- *Streaming* es otro término utilizado específicamente para describir los datos que fluyen constantemente en un sistema informático y que se procesan inmediatamente. Tienen su problemática particular.
- Las analíticas en tiempo real, aportan información muy actual, por lo que los responsables de la organización pueden realizar una toma de decisiones oportuna, preventiva y reactiva.

Big Data Landscape 2016 (Version 3.0)



Last Updated 3/23/2016

FIRSTMARK