Universidad ORT Uruguay

Facultad de Ingeniería

Mapeo sistemático y evaluación de arquitecturas de software para contextos de big data

Entregado como requisito para la obtención del título de

Master en Ingeniería

Juan Pablo Russo – 126376

Tutor: Martín Solari

2018

Declaración de autoría

Yo, Juan Pablo Russo, declaro que el trabajo que se presenta en esta obra es de mi propia mano. Puedo asegurar que:

- La obra fue producida en su totalidad mientras realizaba la Maestría en Ingeniería;
- Cuando he consultado el trabajo publicado por otros, lo he atribuido con claridad;
- Cuando he citado obras de otros, he indicado las fuentes. Con excepción de estas citas, la obra es enteramente mía;
- En la obra, he acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, he explicado claramente qué fue contribuido por otros, y qué fue contribuido por mí;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

Juan Pablo Russo – 20/07/2018

Abstract

Big data es la información caracterizada por un volumen, velocidad y variedad alta de datos que requieren métodos analíticos y tecnologías específicas para poder ser gestionados y transformados en valor agregado para el usuario. El mercado de servicios de big data ha comenzado a crecer sostenidamente en los últimos años. Sin embargo, su rápido crecimiento trae varios desafíos a superar para la ingeniería de software. Las arquitecturas de software se vuelven relevantes en este contexto donde los estilos y patrones tradicionales no son suficientes para el diseño y desarrollo de software.

Esta tesis tiene como objetivo explorar los desafíos y prácticas utilizadas durante el proceso de diseño arquitectónico en contextos de big data. En primer lugar, se realizó un mapeo sistemático de la literatura para identificar y categorizar propuestas de arquitecturas de software. Luego se profundiza la evaluación de dichas arquitecturas para identificar, describir y discutir el impacto de un conjunto de tácticas arquitectónicas sobre los atributos de calidad propios de big data.

Se concluye que existen una variedad de propuestas de arquitectura de software industriales, teóricas y de referencia para big data. Estas propuestas muchas veces difieren en las capas y la separación de responsabilidades, por lo que dificulta al practicante diseñar una solución que se adapte a su contexto de uso. Por otra parte, los resultados del análisis de estas arquitecturas indican la existencia de requerimientos complejos, similares a los encontrados en sistemas distribuidos, pero a mayor escala, determinados por las características de gran volumen, variedad y velocidad de datos. Estos resultados muestran la oportunidad de buscar mejoras al proceso del diseño arquitectónico, adoptando prácticas como el uso de tácticas de arquitectura, para capturar las decisiones de diseño propias de big data.

Palabras clave

arquitecturas de software; big data; mapeo sistemático; atributos de calidad; tácticas arquitectónicas.

Índice

Índice de Tablas

Índice de Figuras

1. Introducción

1.1. Desafíos de Ingeniería de Software en Big Data

En los últimos años, el continuo incremento en el volumen de datos generados mundialmente ha sido amplificado por el surgimiento de las redes sociales, así como la interconectividad impulsada por el Internet of Things (IoT) y la utilización de una variedad de formatos de datos [1]. En el año 2016, se estima que se generan 2.5 quintillones de bytes de datos diariamente, siendo el 90% de los datos mundiales actuales creados en los últimos dos años [2]. En 2001, Laney define las características de big data con el desarrollo del modelo de 3Vs (Volumen, Velocidad y Variedad) [3]. Según IDC (International Data Corporation), el mercado de tecnologías y servicios de big data espera crecer a un ratio compuesto anual de 23% hasta alcanzar los USD 58.9 billones en el 2020 [4], mientras que otros estudios predicen llegar a los USD 80 billones hacia finales del 2030 [5].

Sin embargo, existen varios desafíos que el área de big data debe superar para alcanzar dichas perspectivas de crecimiento. En el 2012 se realiza el primer congreso de big data de la IEEE [6], mientras que en el 2015 se desarrolla el primer workshop de ingeniería de software para big data [7]. En dichas conferencias se exponen varios artículos que definen la problemática de aplicar ingeniería de software a los problemas de big data. La industria todavía se ocupa principalmente de los problemas de infraestructura y técnicas de análisis de datos pero carece de madurez en el proceso propiamente de ingeniería de software en big data [8]. Esto genera que existan múltiples desafíos en las actividades de ingeniería de software [9] resumidas a continuación:

- **Ingeniería de requerimientos:** Es difícil lograr las características de una buena especificación ya que los requerimientos de big data incluyen aspectos de sistemas de tiempo real para el procesamiento de datos y a su vez sufren de variabilidad en las entradas y salidas durante el transcurso del tiempo [10] [11] (lo que se conoce como concept drift[1] [12]). Esto deriva en la complejidad de los modelos necesarios para resolver los requerimientos y datos cambiantes en el tiempo [13].

[1] El concept drift es un problema que ocurre con el cambio de significado de un mismo concepto en el tiempo. En general implica que la distribución probabilística de las variables se ve afectada por eventos temporales que son difíciles de predecir. Esto deja obsoleto a los modelos de predicción originales que deben ser actualizados o deben contemplar este tipo de desviaciones en su modelo de datos [185].

- **Diseño y arquitectura**: La velocidad, variedad y volumen de datos, característicos de big data, presentan nuevos desafíos en el diseño de arquitecturas y soluciones en la industria para lograr gestionarlos [14] [15]. También se presentan los desafíos para cumplir con los atributos de calidad [16] y privacidad de los datos de los usuarios [17]. Los atributos propios de los contextos de big data generan la necesidad de tener arquitecturas complejas de varias capas. Esto se puede observar en varias de las arquitecturas de software que contienen una variedad de capas (cada una con sus propios desafíos tecnológicos) para lograr la persistencia, extracción, procesamiento, análisis, transformación y visualización de datos con características de big data. Por lo tanto, es crucial el entendimiento y diseño de cada componente para conseguir una solución integral de software. Esto implica muchas veces un gran desafío para el practicante que debe manejar los diferentes conceptos y tecnologías propias de cada capa.
- **Implementación:** La implementación [18] exige equipos multidisciplinarios con variadas habilidades en la recolección de datos [19] para su posterior análisis y visualización [20]. La naturaleza distribuida de las soluciones de big data implica varios desafíos técnicos de integración e infraestructura para su puesta en operación y mantenimiento [21] de forma de mantener los atributos de calidad como la escalabilidad, disponibilidad, y performance adecuadas.
- **Pruebas:** La verificación de los algoritmos utilizados para analizar y predecir los resultados de los datos recolectados usualmente carece de un oráculo para definir los resultados esperados bajo el problema ya discutido de concept drift. También el gran volumen de datos hace impráctico la verificación manual de resultados. Esto implica la necesidad de aplicar técnicas como metamorphic testing [22] que identifican las relaciones y probabilidades en el modelo que permitan evitar el uso de un oráculo.

1.2. Motivación

Esta tesis se encuentra motivada por la complejidad y dificultad inherente en el diseño arquitectónico de soluciones de big data, así como la carencia de elementos para capturar y aplicar decisiones de diseño por los arquitectos de software en dichos contextos. El incremento en la generación de datos globales hace inminente la llegada de los desafíos de big data a la industria de software. El desconocimiento de los atributos de calidad y prácticas apropiadas para el diseño arquitectónico genera incertidumbre para los practicantes sobre la facilidad de generar soluciones de software de alta calidad bajo contextos de big data. Esto se ve reflejado

en que cada organización genera sus propias soluciones para lidiar con las problemáticas de big data, sin contar con una guía clara o estandarizada para trabajar en este tipo de contextos. A continuación se resumen algunos de los desafíos propios de las características de big data que tienen un impacto sobre las arquitecturas de software.

- **Volumen**: Se estima que 2.5 quintillones de bytes de datos son generados en forma diaria y que el 90% de la información de la humanidad se genera en los últimos 2 años [1]. Por lo tanto, los sistemas deben poder escalar para lograr almacenar y procesar dicho volumen de datos, que se pronostica doblará su cantidad cada dos años.
- **Variedad**: Las redes sociales, el Internet of Things y el incremento del uso dispositivos con diferentes sensores ha provocado una variedad de formatos no estructurados de datos. Se estima que los datos no estructurados como imágenes, videos, audio y texto constituyen cerca del 95% de los datos generados en la actualidad [42]. Esta diversidad de formatos y estructuras de datos imponen requerimientos en el diseño del almacenamiento de información que no pueden ser resueltos con base de datos relacionales tradicionales.
- **Velocidad**: Cada vez más las organizaciones deben poder generar valor en su toma de decisiones a través del análisis en tiempo real de sus datos. Un ejemplo son las organizaciones financieras que deben poder analizar las transacciones rápidamente para evitar eventos fraudulentos. Esto implica que la aproximación tradicional de Business Intelligence, con procesamiento en lotes y el uso de datawarehouses, no es suficiente y se debe poder analizar un flujo constante de datos en tiempo real.

1.3. Objetivo de la tesis

El objetivo general de la tesis es entender los desafíos al diseñar soluciones integrales de software, bajo contextos de big data, desde un punto de vista pragmático para un arquitecto de software. Esto significa entender los elementos y conocimiento necesarios para poder identificar, diseñar y evaluar arquitecturas de software de big data, en sus diferentes contextos de uso, a nivel industrial.

Para poder organizar el proceso de investigación se definen dentro de la tesis tres objetivos específicos:

1. Identificar los desafíos en el diseño de arquitecturas de software para contextos complejos de big data. Esto permitirá comprender los factores que dificultan el diseño de soluciones con características de grandes volúmenes, velocidad y variedad de datos. Por otro lado, permitirá entender del estado actual de las propuestas y diseños de arquitectura de software aplicados por distintas organizaciones. Para concretar este objetivo se utilizará la metodología de estudio de mapeo sistemático de la literatura.
2. Analizar las prácticas y técnicas de arquitectura, utilizadas actualmente en la industria, en el contexto particular de las soluciones de big data. Por un lado, se buscará confirmar que los escenarios de big data traen problemáticas particulares que las tecnologías tradicionales no son capaces de resolver. También se buscará obtener un análisis de los atributos de calidad impuestos sobre las arquitecturas de software en big data.
3. Elaborar guías para mejorar la adopción y utilización de prácticas de arquitectura apropiadas en el momento de diseñar soluciones de big data. Dichas guías surgirán de consolidar los resultados de los estudios realizados. Las mismas incluirán tácticas y patrones de arquitectura adecuadas para facilitar las decisiones durante el proceso del diseño arquitectónico.

1.4. Contribuciones de la tesis

Las siguientes serán las principales contribuciones de esta tesis:

1. **Identificación de propuestas de arquitectura de software aplicadas actualmente en sistemas de big data.**

   Del mapeo sistemático de la literatura se elaborará una clasificación de las propuestas de arquitectura de software de diferentes organizaciones y comunidades de arquitectos. Esta clasificación permitirá sintetizar el conocimiento de los diferentes diseños de arquitectura de software en big data para conocer el estado del arte en esta área del conocimiento.
2. **Análisis sobre las prácticas actuales para diseñar aspectos de big data.**

   Del procesamiento de los resultados del mapeo sistemático se recolectarán los diferentes tipos de arquitectura de software y sus contextos de aplicación. Dentro del estudio se podrá identificar la cobertura de las arquitecturas de software en las diferentes capas de la aplicación y el detalle del tratamiento del diseño de cada una. Dicha evidencia confirmará los desafíos y complejidades inherentes al diseñar soluciones de big data y el estado actual de las prácticas de arquitectura de software utilizadas.

3. **Identificación de atributos de calidad y escenarios de uso en contextos de big data.** Se elaborará una lista de atributos de calidad encontrados durante el estudio de la literatura que hayan sido planteados por distintos autores como posibles requerimientos necesarios en contextos de big data. Se utilizarán escenarios de uso comunes para ejemplificar más detalladamente cada atributo de calidad. Esto permitirá plantear los requerimientos particulares de big data e identificar los desafíos al diseñar arquitecturas de software.
4. **Discusión sobre tácticas y patrones de arquitectura para guiar las decisiones de diseño.** Se elaborarán argumentos en base a los atributos de calidad y escenarios de uso, buscando profundizar el conocimiento sobre los requerimientos no funcionales específicos en contextos de big data. En esta discusión se plantearán recomendaciones para guiar al practicante durante el proceso del diseño arquitectónico. Dichas sugerencias incluirán tácticas y patrones de arquitectura necesarios para satisfacer los atributos de calidad anteriormente identificados.

1.5. Estructura de la tesis

El capítulo 2 comienza realizando una serie de definiciones necesarias para la correcta comprensión de los temas tratados en el resto de la tesis. El resto del capítulo 2 presenta el estado del arte relacionado con las arquitecturas de software en big data, mostrando el marco conceptual en el que se desarrolla la investigación, problemas y oportunidades.

El capítulo 3 describe el método general de investigación detallando el flujo de trabajo que se seguirá. Además, se presentan los dos principales estudios que se llevarán a cabo junto con la motivación de estos y la descripción del método a seguir.

El capítulo 4 presenta el mapeo sistemático de la literatura realizado para estudiar y clasificar las arquitecturas de software en big data. En este capítulo se detalla la preparación del estudio, los resultados de este y la correspondiente discusión de ellos.

El capítulo 5 describe el estudio del análisis de las prácticas de arquitectura de software en big data. En este capítulo se describen los atributos de calidad y escenarios de uso identificados, los resultados obtenidos del análisis y la discusión realizada sobre ellos.

En el capítulo 6 se lleva a cabo el último paso del proceso de investigación que consiste en una discusión general acerca de los resultados obtenidos en ambos estudios. Dicha discusión se

encuentra alineada a los objetivos específicos que llevarán a la concreción del objetivo general planteado.

Finalmente, el capítulo 7 presenta un resumen de la investigación, principales resultados y aportes obtenidos, publicaciones asociadas al trabajo de tesis y futuras líneas de investigación.

2. Estado de la cuestión

Existen varios conceptos que se utilizan frecuentemente en la investigación de las arquitecturas de software en contextos de big data. Este capítulo comienza definiendo los principales términos para una correcta comprensión de los aportes de la tesis. Como parte del estado de la cuestión se describen las principales características de las arquitecturas de software que dan fundamento a los hallazgos de la tesis. Se describe el trabajo previo relacionado con la línea de investigación con un resumen final del estado de la cuestión.

2.1. Definiciones

2.1.1. Big data

El primer uso del término de big data se remonta a la década de los 90s por investigadores de la NASA mientras construían interfaces para simuladores de vuelo. *"[D]ata sets are generally quite large, taxing the capacities of main memory, local disk, and even remote disk [...]". "We call this the problem of big data."* [23]

Una de las definiciones más contemporáneas del término es definida en el estudio McKinsey sobre big data en 2011 [24]. *"Big data refers to datasets whose size is beyond the ability of typical database software tools to capture, store, manage, and analyze."*

Recientes artículos [25] [26] han tratado de consolidar la definición de big data y sus términos relacionados. Los autores consolidan las varias definiciones de big data en la literatura [27] bajo 4 dimensiones: información, tecnología, métodos e impacto. *"Big Data is the Information asset characterized by such a High Volume, Velocity and Variety to require specific Technology and Analytical Methods for its transformation into Value."*

2.1.1.1. Información

El modelo DIKW (Data-Information-Knowledge-Wisdom) explica las diferencias existentes entre datos, información, conocimiento y sabiduría, así como las transformaciones cognitivas necesarias para relacionar la jerarquía [28] como muestra la Figura 1.

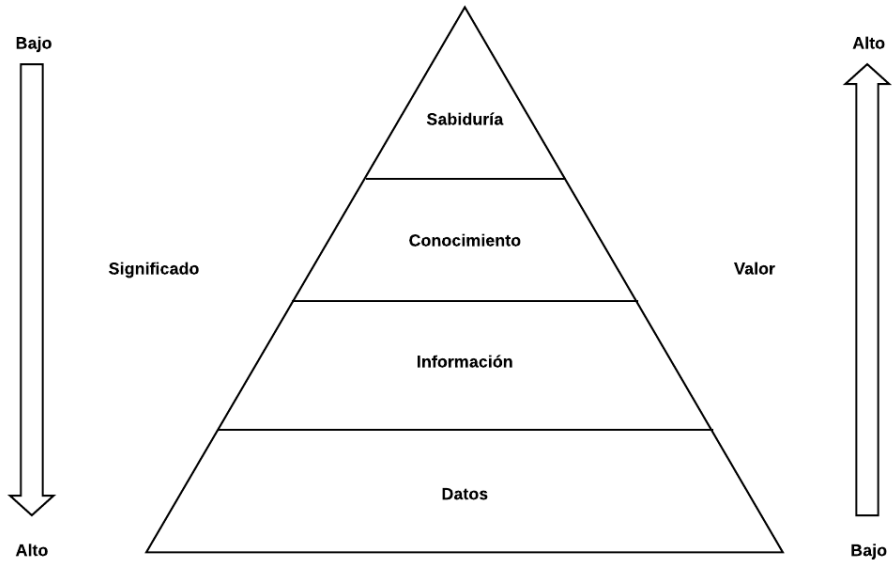

Figura 1 Jerarquía de datos, información, conocimiento y sabiduría

Para lograr las transformaciones dentro de la jerarquía DIKW es importante estudiar las características particulares de los datos, en un contexto de big data, descriptos en el modelo original de 3Vs (volumen, velocidad y variedad) [3][2].

La característica de volumen es la referida como la más importante y distintiva dentro de big data [29]. El gran volumen de datos puede facilitar la generación de modelos para el análisis [30], siempre que se cuide la calidad de los datos [31] [32]. Sin embargo, también traen nuevos desafíos para su almacenamiento y manipulación [16]. El teorema de CAP (Consistency, Availability, Partitioning) [33] demuestra las problemáticas de disponibilidad [34] y consistencia [35] al tener que particionar horizontalmente [36] los grandes volúmenes de datos en big data.

La velocidad refiere a la frecuencia en el flujo de datos [37] generados en escenarios distribuidos. Existen dos tipos de flujos de datos principales dentro de las soluciones de big data.

1. El flujo de datos generados por una variedad de fuentes heterogéneas que deben ser integradas al conjunto de datos ya existentes [38].

[2] Existen varias extensiones al modelo de 3Vs [29][186][187] que agregan otras dimensiones como Veracidad, Valor, Visibilidad.

2. El flujo de datos procesados y analizados necesarios para resolver y visualizar las consultas de los usuarios [39].

La variedad consiste en la diversidad de fuentes y estructuras de almacenamiento. Esta diversidad se puede categorizar de la siguiente forma:

- Datos estructurados que generalmente tienen modelos asociados que permiten definir su estructura y significado [40].
- Datos desestructurados como imágenes, videos, audio y texto que constituyen cerca del 95% de los datos generados hoy en día [41].
- Datos semi-estructurados que contienen la información asociada a la estructura en los propios datos, lo que se conoce como "self-describing" [42]. En general son datos que no tienen campos fijos pero contienen tags o marcadores que permiten separar los elementos de datos, como lo son los formatos XML o HTML [43].

2.1.1.2. Tecnología

Las tecnologías tradicionales no son suficientes para el procesamiento de big data, por lo tanto se proponen tecnologías y técnicas alternativas para su solución [44]. Fischer [45] sugiere que el concepto del volumen de datos, en términos de tamaño, está relacionada con la ley de Moore [46], y consecuentemente con la capacidad actual de los almacenamientos comerciales. Por lo tanto en big data, la extensión y variedad del conjunto de datos, así como la complejidad de las operaciones necesarias para su procesamiento, implican un riguroso uso de memoria y requerimientos específicos de performance [26]. La naturaleza distribuida de los datos requiere tecnologías especiales para lograr transmitir grandes cantidades de información y lograr monitorear la eficiencia del sistema utilizando técnicas de benchmarking específicas [47].

Esto implica que nuevas tecnologías son necesarias para la gestión de big data. Para el almacenamiento de grandes volúmenes es necesario utilizar sistemas de archivos [48] [49] o base de datos [50] [51] distribuidas. Este nuevo paradigma utiliza base de datos denominadas NoSQL [52] que abarcan estructuras de valor-clave, familia de columnas, documentales y de grafos. El procesamiento de datos se puede realizar en lotes o tiempo real [53]. Muchas de estas tecnologías buscan paralelizar la ejecución con nuevos modelos de programación como Map Reduce [54].

2.1.1.3. Métodos

El análisis y transformación de grandes cantidades de datos para obtener el valor agregado para los negocios requiere métodos adicionales a las estadísticas tradicionales. Existen artículos [24] [55] que enumeran la variedad de métodos disponibles para aplicar en diferentes contextos. Los métodos pueden ser clasificados según su estilo de aprendizaje [56]:

- **Aprendizaje supervisado***:* Se tiene un conjunto de datos de entrenamiento que se pueden clasificar y conocer los resultados esperados de su análisis. De esta forma el modelo de predicción se genera a través de un proceso de entrenamiento que continúa hasta conseguir resultados con la precisión deseada.
- **Aprendizaje no supervisado***:* Los datos de entrada no están clasificados ni se conocen sus resultados esperados. Por lo tanto, el modelo de predicción es generado deduciendo las estructuras presentes en los datos de entrada. Este proceso debe inducir reglas y patrones generales a través de un proceso de organización y clasificación de los datos (e.g. clustering).
- **Aprendizaje semi-supervisado***:* Existen datos de entrada que pueden ser clasificados pero otros deben ser deducidos por el propio entrenamiento. Por lo tanto, el modelo de predicción debe poder aprender las estructuras necesarias para clasificar los datos de forma de poder realizar las predicciones.

2.1.1.4. Impacto

La transformación de datos en ventaja competitiva es lo que está revolucionando el área de big data en el mundo empresarial [57]. La naturaleza universal de la actual producción de información y su disponibilidad permite la proliferación de aplicaciones en variados campos y sectores de la industria. Por ejemplo, el análisis y correlación de búsquedas en Google fueron utilizados tanto para predecir epidemias de gripe [58], así como las cifras de desempleo [59] e inflación [60].

Sin embargo, también existen aspectos negativos como la preocupación por la privacidad de los datos o la accesibilidad de la información [61]. Las grandes corporaciones suelen restringir el acceso al conocimiento creando barreras de entrada al mercado para la competencia [62].

2.1.2. Arquitectura de Software

Los sistemas de software son construidos para satisfacer las necesidades de negocio de las organizaciones. La arquitectura de software permite diseñar, analizar y documentar los elementos de software, sus relaciones y propiedades que al ser implementadas permiten satisfacer dichas necesidades de negocio.

El SWEBOK define la arquitectura de software como: *"The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both"* [63]

Los siguientes conceptos permiten extender la definición y comprensión de la arquitectura de software.

2.1.2.1. La arquitectura es un conjunto de estructuras de software

La estructura se conforma por un conjunto de elementos de software con sus relaciones y propiedades. Los sistemas contienen múltiples estructuras, muchas veces agrupadas en diferentes vistas [64]. La descripción de las decisiones que explican como dichas estructuras logran satisfacer el comportamiento esperado del sistema se define como el rationale [65].

Hay tres categorías de estructuras de software, que juegan un rol importante al diseñar, documentar y analizar la arquitectura:

1. **Estructuras de módulos:** Son estructuras estáticas que permiten particionar el sistema en unidades de implementación. Existen distintos niveles de estructuras como pueden ser las clases o componentes que se agrupan en estructuras más generales como son los módulos o paquetes, que a su vez se pueden agrupar en las diferentes capas del sistema.
2. **Estructuras de componentes y conectores:** Son estructuras dinámicas que describen la interacción en tiempo de ejecución de los componentes de software para lograr las funciones requeridas del sistema.
3. **Estructuras de asignación:** Describen la relación entre las estructuras de software con los elementos externos del ambiente donde el software finalmente se instalará y ejecutará.

2.1.2.2. La arquitectura es una abstracción

La arquitectura es una abstracción del sistema que describe cierta parte de su estructura, omitiendo ciertos detalles irrelevantes para el modelado bajo diferentes puntos de vista [66]. En todos los sistemas modernos, los elementos de software interactúan entre sí a través de interfaces que dividen la definición del componente en aspectos públicos y privados. La arquitectura se centra en el estudio de los aspectos públicos de los componentes, para estudiar sus interrelaciones, composiciones e interacciones, evitando describir detalles de implementación interna de sus componentes constitutivos [67].

2.1.2.3. Arquitectura de referencia de software

Las arquitecturas de referencia de software presentan una forma de resolver el problema de gestionar la complejidad, tamaño y alcance de las arquitecturas bajo los diferentes contextos sobre los que son utilizados [68]. La siguiente definición permite entender su objetivo: *"Las arquitecturas de referencia capturan la esencia de las arquitecturas existentes, para incorporar la visión de las necesidades y evoluciones futuras de forma de proveer una guía en la asistencia de posteriores diseños y desarrollos de sistemas."* [69]

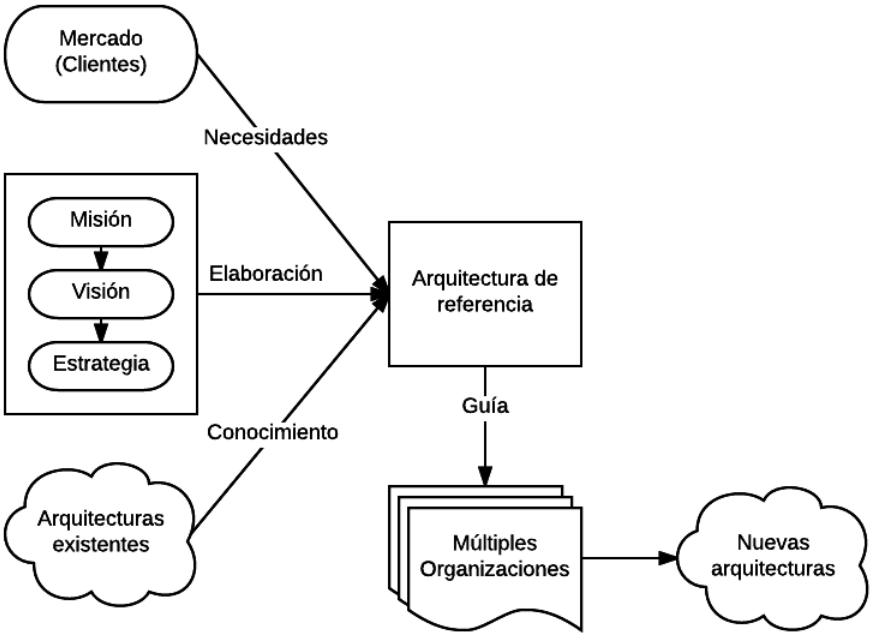

Figura 2 Resumen del rol de las arquitecturas de referencia

Las arquitecturas de referencia capturan el conocimiento de arquitecturas existentes, contemplando las futuras necesidades del mercado y se basan en la elaboración de una misión, visión y estrategia de uso. Estas arquitecturas de referencia proveen una guía para múltiples organizaciones dentro del mismo dominio de aplicación para crear y evolucionar sus arquitecturas como muestra la Figura 2.

Bass et al. [67] define las arquitecturas de referencia a partir de su relación con los patrones de arquitectura y modelos de referencia como muestra la Figura 3.

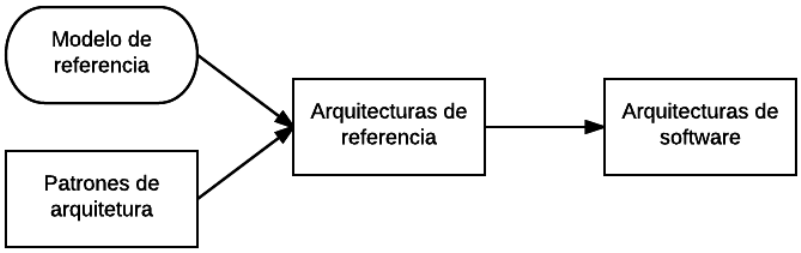

Figura 3 Relación entre patrones, modelos y arquitecturas de referencia

1. **Patrón de arquitectura:** Es una descripción de los tipos de elementos del sistema y sus relaciones sobre un conjunto de restricciones de cómo pueden ser utilizados. Estas

restricciones sobre la arquitectura definen y describen un conjunto o familias de arquitecturas que los satisfacen. Uno de los aspectos más importantes de los patrones es que estas restricciones logran satisfacer ciertos atributos de calidad esperados.

2. **Modelo de referencia:** Un modelo de referencia es una división de la funcionalidad del sistema con su flujo de datos asociado. Es una descomposición estándar de un problema conocido en sus partes constitutivas que cooperativamente resuelven el problema original. Es necesario la madurez de conocimiento sobre el dominio de aplicación para lograr definir dicho modelo canónico.
3. **Arquitectura de referencia:** Una arquitectura de referencia es un modelo de referencia mapeado sobre los elementos de software que implementan la funcionalidad definida en dicho modelo. Mientras que el modelo de referencia divide la funcionalidad, la arquitectura de referencia es el mapeo de dicha funcionalidad sobre los componentes de software del sistema.

2.1.3. Patrones y tácticas de arquitectura de software

Las características propias de gran volumen, velocidad y variedad en big data hacen que los diseños de arquitectura de software tengan sus propios desafíos que serán explorados durante la tesis. Una forma de visualizar está complejidad es a través de los atributos de calidad y sus escenarios de uso asociados. Los escenarios constan de las siguientes partes principales:

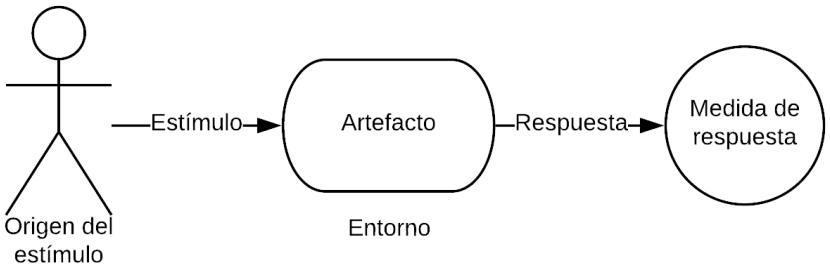

Figura 4 Elementos de un escenario

1. **Origen del estímulo:** Esto representa una entidad (humano, sistema u otro actuador) que genera el estímulo.
2. **Estímulo:** El estímulo es una condición que necesita de una respuesta al ingresar al sistema.

3. **Entorno:** El estímulo ocurre bajo ciertas condiciones del entorno. El sistema puede encontrarse en una condición de sobrecarga u operación normal, o algún otro estado relevante. Representa el modo de ejecución bajo el cual se encuentra el sistema.
4. **Artefacto:** Representa la parte del sistema que es estimulada. Puede ser todo el sistema en su conjunto o ciertos componentes de este.
5. **Respuesta:** La respuesta es la actividad resultante de la llegada del estímulo al artefacto.
6. **Medida de respuesta:** La restricción a un atributo especifico de la respuesta que debe poder cuantificarse para poder medir el cumplimiento del requerimiento.

El arquitecto de software debe tomar decisiones de diseño que logren satisfacer los atributos de calidad propios de los escenarios de uso del sistema. Una buena forma de capturar estas decisiones de diseño es a través de las tácticas y patrones de arquitectura.

Una táctica de arquitectura [67] es una forma de controlar la respuesta del estímulo para satisfacer un atributo de calidad a través de decisiones de diseño de arquitectura. Especifica como los parámetros del modelo del atributo de calidad pueden ser controlados para conseguir una respuesta deseada frente al estímulo del artefacto, como muestra la Figura 5. Esto implica que las tácticas de arquitectura representan conocimiento codificado sobre la relación entre las decisiones de arquitectura y los parámetros de los atributos de calidad. Se pueden utilizar frameworks de razonamiento general [70] (como teoría de colas), para crear modelos concretos (como modelos de programación de tareas) que caractericen los parámetros de los atributos de calidad. Esto permite modelar y entender el impacto de las decisiones de diseño sobre los parámetros de calidad y respuestas deseadas.

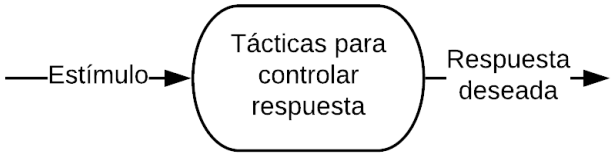

Figura 5 Táctica de arquitectura

El proceso de selección de tácticas [71] comienza con un escenario concreto que determina un escenario más general del sistema. El escenario general contiene parámetros del atributo de calidad que se relaciona con uno (o varios) posibles frameworks generales de razonamiento. Estos marcos de razonamiento contienen un conjunto de parámetros con sus relaciones que permiten describir las propiedades del sistema. Los escenarios concretos especifican valores

necesarios para dichos parámetros que caracterizan las propiedades del sistema. Estos valores son los que se fijan en el modelo para entender que otros parámetros se pueden ajustar a través de decisiones de diseño determinadas por las tácticas de arquitectura.

Los patrones son soluciones a problemas recurrentes como se define en el libro de Pattern Language [72]. "*Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem. In such a way that you can use this solution a million times over, without ever doing it the same way twice*". Por lo tanto un patrón de software describe un problema con su solución genérica asociada para un contexto dado. Los patrones de software pueden ser aplicados a distintos niveles de diseño y codificación. Los patrones de arquitectura describen una estructura de alto nivel del sistema con su comportamiento asociado. Existen varios catálogos de patrones de arquitectura [73] [74] [75] en la literatura.

2.2. Trabajo previo

2.2.1. Estandarización

El NIST (National Institute of Standards and Technology, de Estados Unidos) propuso un grupo de trabajo (NBD-PWG) [76] para crear un framework extensible de big data con nueve publicaciones. Estas publicaciones definen taxonomías [77], arquitecturas de referencia [78], casos de uso [79] y un roadmap [80] general que busca la estandarización de prácticas en contextos de big data.

ISO (International Organization for Standardization) por su parte genera un reporte preliminar [81] que define términos y conceptos relevantes de big data. Este reporte presenta una descripción del estado actual del trabajo y prioridades de estandarización necesarias para la comunidad de big data.

2.2.2. Comunidades de arquitectura de software

El SEI (Software Engineering Institute) [16] presenta un estudio de los desafíos de arquitectura de software a nivel de datos, distribución y despliegue de soluciones de big data. En el artículo se discuten, lo que los autores entienden como, los tres principales atributos de calidad bajo contextos de big data: eficiencia, disponibilidad y escalabilidad. El estudio también provee la

descripción de tácticas de arquitectura en cada uno de los niveles con respecto a los atributos de calidad anteriormente mencionados.

Dentro del SEI, Gorton [82] explica las dificultades de diseñar sistemas de gestión de big data altamente escalables y distribuidos. El estudio plantea una taxonomía que captura las características de las arquitecturas de big data más relevantes en la capa de datos. La taxonomía permite a su vez evaluar y comparar plataformas de base de datos distribuidas con una herramienta denominada QuABaseDB [83].

Por su lado Kazman [84] plantea una aproximación ágil de prototipado para el diseño de sistemas de big data. El proceso denominado BDD (Big Data Design) contiene una serie de fases, desde el descubrimiento del valor del producto hasta el diseño de la solución, utilizando arquitecturas de referencia y un catálogo de conceptos de diseño. Este proceso es extendido para el diseño de analíticas de big data denominado Architecture-centric Agile Big data Analytics (ABBA) [85].

Durante los años recientes surgen arquitecturas de referencia [86], [87], [88], [89], [90] que tratan de abstraer y capturar el conocimiento de las arquitecturas teóricas y concretas de la industria. Lo valioso de estas propuestas es que analizan arquitecturas existentes buscando generar una abstracción común que permita representar las arquitecturas que los autores interpretan como importantes. Estas arquitecturas de software serán analizadas como parte de la tesis.

2.2.3. Revisiones sistemáticas y surveys

También existen en la literatura revisiones sistemáticas relacionadas con el área de big data. Una de ellas sintetiza todos los artículos relacionados con Hadoop [91] como fuente para los practicantes que desean saber más en detalle la evolución empírica de dicho stack tecnológico. Por otro lado, existe una revisión sistemática que estudia el contexto de big data en soluciones de cloud computing [92], encontrando varios desafíos a nivel de almacenamiento con respeto a la atomicidad y durabilidad de los datos. También existe una revisión sistemática relacionada con el nuevo campo de software analytics [93], en el cual los autores concluyen que la información del análisis del software sirve para desarrolladores, testers y gestores de proyecto, pero no hay mención directa de los arquitectos de software dentro del campo de estudio.

Por otro lado, existen varios surveys sobre distintos aspectos de big data como es el procesamiento en tiempo real [94], la programación de tareas [95], [96], herramientas para el análisis de datos [97] y la eficiencia en el almacenamiento de datos [98].

2.3. Resumen del estado de la cuestión

Existe un número creciente de arquitecturas de software en contextos de big data que son publicadas por organizaciones o comunidades de arquitectura que demuestra el interés general de la industria. La elaboración de este estado de la cuestión permitió identificar los siguientes huecos en el conocimiento:

- Aunque se encontraron distintas revisiones sistemáticas y surveys que describen e incorporan información valiosa de big data para ser utilizada por los practicantes, no se encontró ninguna fuente organizada ni con comparativa rigurosa sobre las arquitecturas de software de big data.
- Si bien existen las 3Vs (Volumen, Variedad y Velocidad) para describir las características de big data, se carece de un compendio de escenarios de uso que permitan determinar los atributos de calidad comunes de big data. Este compendio permitiría a los practicantes e investigadores disponer de una mayor información para discernir sobre los requerimientos no funcionales bajo contextos de big data.
- Existen una variedad de arquitecturas de software que describen a alto nivel las posibles capas necesarias en contextos de big data. Sin embargo, se requiere enriquecer el cuerpo de conocimiento con estilos y tácticas de arquitectura que ayuden al practicante en el diseño e implementación de sus soluciones de big data.

3. Metodología de investigación

El método general de investigación se diseña alineado con los objetivos específicos de la tesis. Podemos detallar la pregunta de investigación con el paradigma de GQM (Goal-Question-Metric) [99]. Para esto se definirá un objetivo al nivel conceptual para generar preguntas a nivel operacional que tendrán respuesta por la tesis con el estudio de ciertos elementos definidos a nivel cualitativo del modelo GQM como muestra la Figura 6.

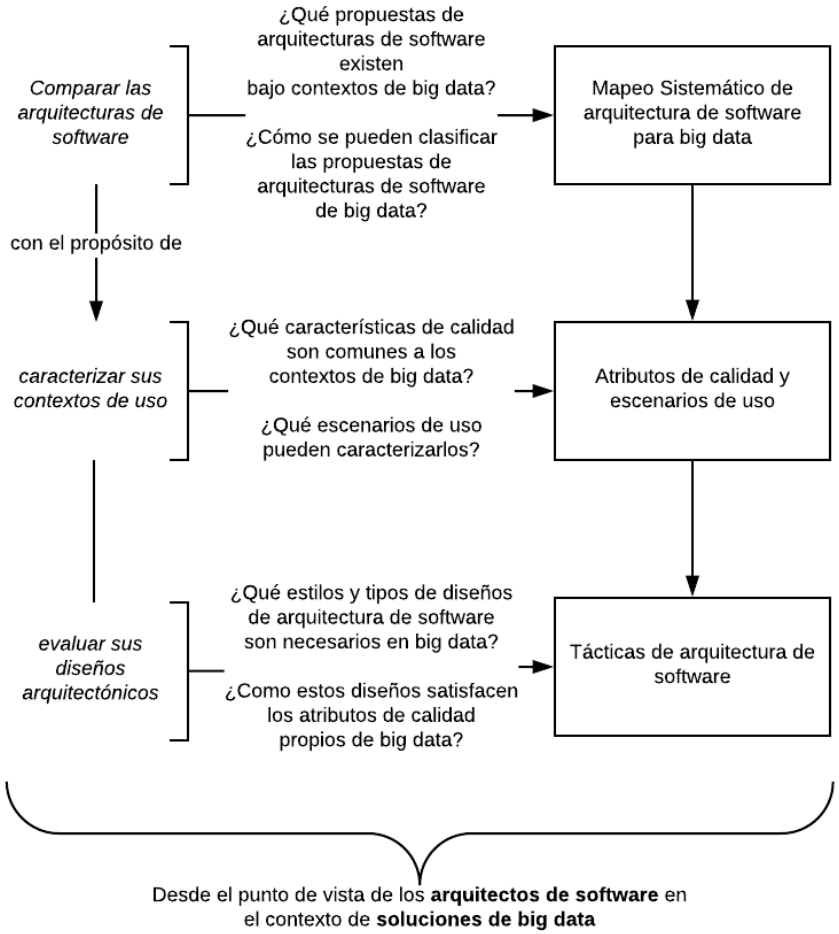

Figura 6 Modelo general de investigación

3.1. Preguntas de investigación

La objetivo principal del nivel conceptual alineado con los objetivos de la tesis puede ser planteado utilizando la plantilla sugerida por Basili [100].

Comparar las ***arquitecturas de software*** con el propósito de ***caracterizar los contextos de uso*** y ***evaluar sus diseños arquitectónicos*** con respecto a las ***tácticas de arquitectura*** disponibles desde el punto de vista de los ***arquitectos de software*** en el contexto de ***soluciones de big data***.

La definición del nivel conceptual permite generar una sería de preguntas que deberán ser respondidas para alcanzar el objetivo planteado. Estas son:

- **Q1**: ¿Qué propuestas de arquitecturas de software existen bajo contextos de big data?
- **Q2**: ¿Cómo se pueden clasificar las propuestas de arquitecturas de software de big data?
- **Q3**: ¿Qué características de calidad son comunes a los contextos de big data?
- **Q4**: ¿Qué escenarios de uso pueden caracterizar los contextos de big data?
- **Q5**: ¿Qué estilos y tipos de diseños de arquitectura de software son necesarios en implementaciones de big data?
- **Q6**: ¿Cómo estos diseños satisfacen los atributos de calidad propios de big data?

Parte del objetivo de la tesis es encontrar elementos en el nivel cualitativo que permitan responder las preguntas planteadas. El primer paso busca identificar y clasificar las propuestas de arquitectura de software para big data. Para esto se decide establecer una línea base realizando un mapeo sistemático de la literatura sobre los tipos de arquitectura de software existentes en contextos de big data. Ya que no existe una estandarización establecida de big data se espera encontrar una diversidad de propuestas para resolver los desafíos de diseño arquitectónico. Será parte del estudio entender las similitudes y diferencias de las propuestas encontrados para lograr clasificarlas y compararlas.

El segundo paso busca entender más detalladamente los contextos de uso para identificar los atributos de calidad relevantes comunes en big data. Estas características permitirán generar escenarios de uso para establecer los requerimientos no funcionales que tengan un impacto en la arquitectura de software de los sistemas. Se espera encontrar características comunes en las diferentes propuestas de arquitectura de software relevadas y clasificadas en el primer paso.

El paso final consiste en discutir los hallazgos realizados en los estudios anteriores y elaborar una serie de guías para mejorar la calidad de los procesos de diseño arquitectónico en contextos de big data. Dichas sugerencias serán realizadas a partir de tácticas de diseño identificadas en las arquitecturas de software clasificadas anteriormente. Todos los resultados intermedios de la

tesis pueden ser tomados por otros investigadores para realizar nuevos estudios y por los practicantes para tomar decisiones acerca de las prácticas a utilizar.

3.2. Mapeo sistemático de arquitecturas de software para big data

Los estudios de mapeo sistemáticos de la literatura son un método de investigación que brinda una visión general y tendencia sobre un área de conocimiento. Los estudios de mapeo son de utilidad para relevar, clasificar y estructurar el nivel de evidencia empírica disponible sobre el área de investigación permitiendo ahorrar tiempo y esfuerzo a los investigadores. Esta metodología es utilizada frecuentemente como base para profundizar en trabajos de investigación futuros [101].

3.2.1. Motivación

El mapeo sistemático se realiza para enriquecer el cuerpo de conocimiento con un compendio de las arquitecturas de software de big data. El estudio de mapeo sistemático busca identificar y clasificar un conjunto de estudios primarios que describan propuestas de arquitecturas de software en contextos de big data.

Particularmente, los objetivos de este estudio son:

- Dar una visión general de la evidencia empírica e interés existente en la literatura acerca de las arquitecturas de software de big data.
- Identificar y clasificar el diverso espacio de arquitecturas de software que describan diferentes aspectos de las capas de aplicación.

3.2.2. Método

Para la definición y ejecución del estudio de mapeo sistemático se utilizaron las guías propuestas por Kitchenham y Charters [102] y las recomendaciones actualizadas por Petersen [103] [104].

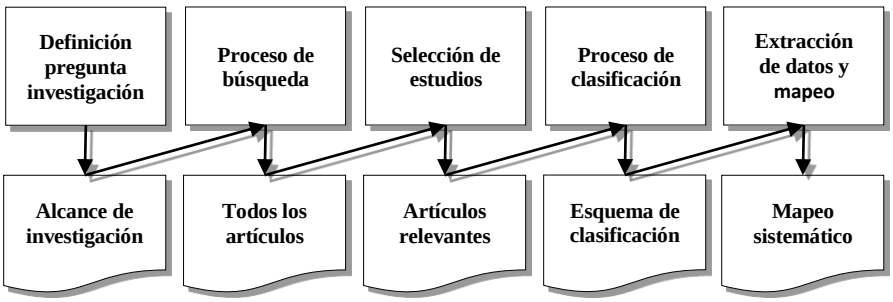

Figura 7 Proceso del mapeo sistemático

Los pasos definidos son los siguientes:

1) **Definir las preguntas de investigación**: Las preguntas de investigación definen el alcance de la investigación y son definidas de acuerdo con los objetivos principales de la tesis.
2) **Proceso de búsqueda**: Se determina las fuentes de datos a utilizar que ofrezcan una mejor cobertura para el área de estudio. Para determinar la estrategia de búsqueda, se utilizan términos de búsqueda relacionados con el área de investigación que son refinados en sucesivas búsquedas exploratorias.
3) **Selección de estudios**: En este paso se definen los criterios de inclusión y exclusión que son usados para incluir los estudios relevantes de forma de responder las preguntas de investigación.
4) **Proceso de clasificación**: Una vez obtenidos los artículos se clasifican según cierta taxonomía definida a los efectos de contestar las preguntas planteadas. Dicha clasificación se realiza tomando el título y resumen de los artículos seleccionados para extraer palabras claves y conceptos que reflejen la contribución del artículo.
5) **Extracción de datos y mapeo**: Finalmente se realiza la extracción de los datos, con el esquema de clasificación, para realizar el mapeo y obtener los resultados del estudio.

3.3. Evaluación de arquitecturas de software para big data

La evaluación de arquitectura de software es una técnica o método que determina las propiedades, fortalezas y debilidades de la arquitectura de software con respecto a los requerimientos funcionales y no funcionales del sistema. Existen varios métodos de evaluación que son clasificados en diferentes estudios [105] [106] como son de experiencia, escenarios, modelos matemáticos y simulación. Las evaluaciones basadas en experiencia utilizan

experiencias y conocimientos pasados de arquitectos y consultores [107]. En las simulaciones se implementa un prototipo de la arquitectura a alto nivel para ser evaluada. Los modelos matemáticos permiten evaluar los requerimientos no funcionales, como eficiencia y disponibilidad, a través de pruebas y métodos matemáticos [108]. Las basadas en escenarios proveen perfiles de escenarios de uso para describir y evaluar cada atributo de calidad [109]. Esta última aproximación será la utilizada para la tesis por aportar una forma descriptiva de interpretar cada atributo de calidad y contar con métodos de evaluación maduros y probados dentro de la industria.

3.3.1. Motivación

El objetivo de una evaluación de arquitectura de software es poder determinar que el diseño de una arquitectura de software cumpla con los requerimientos del sistema. La arquitectura puede ser evaluada en diferentes etapas del proceso de desarrollo de software. En etapas tempranas del diseño, las arquitecturas pueden ser comparadas sobre sus alternativas de diseño para analizar sus fortalezas y debilidades.

El objetivo de este estudio es realizar una evaluación del uso de prácticas y técnicas de arquitecturas de software relevadas en los artículos encontrados durante el mapeo sistemático. Este objetivo general se divide en dos objetivos intermedios:

- Identificar el contexto de uso de cada propuesta de arquitectura de software. Para esto se propondrán escenarios de uso que permitan especificar los atributos de calidad propios de su contexto de uso.
- Seleccionar una serie de tácticas de arquitectura que ayuden a capturar las decisiones de diseño durante el proceso de arquitectura. Esto permitirá entender y evaluar el impacto de cada decisión de diseño sobre los atributos de calidad propios de big data.

3.3.2. Método de evaluación

Los métodos basados en escenarios [110] [111] [112] evalúan la arquitectura con respecto a un conjunto de escenarios de uso. Estos métodos ofrecen una forma sistemática de investigar la arquitectura utilizando escenarios para evaluar si satisfacen sus atributos de calidad. A partir de dicho análisis se pueden comprender el tipo de tácticas de arquitectura a utilizar para satisfacer los escenarios de uso del sistema.

Existen una variedad de métodos de evaluación basado en escenarios como son: architecture trade-off analysis method (ATAM) [111] [113], scenario-based architecture analysis method (SAAM) [114], SAAM for complex scenarios (SAAMCS), architecture level modifiability analysis (ALMA) [115], architecture level prediction of software maintenance (ALPSM) [116], scenario-based architecture reengineering (SBAR) [117], extending SAAM by integration in the domain (ESAAMI) [118], y software architecture comparison method (SACAM) [119].

ATAM es un método de evaluación creado por el SEI para evaluar la aptitud de los diseños de las arquitecturas de software para satisfacer múltiples atributos de calidad en forma simultánea. Este método permite razonar sobre el impacto de las decisiones de arquitectura en la interacción de cada atributo de calidad. Es un modelo en espiral de diseño que permite evaluar y mitigar riesgos sobre las alternativas y decisiones de diseño a nivel arquitectónico. El modelo está originalmente pensado para evaluar arquitecturas de software sobre un dominio de aplicación particular de negocio. En este estudio se adaptan los pasos para generalizar el dominio de aplicación a big data y evaluar las arquitecturas de software relevadas durante el mapeo sistemático. Las etapas generales del modelo siguen siendo conceptualmente las mismas[3], con adaptaciones que se describen a continuación.

1. **Recolección de requerimientos y atributos de calidad**.
   En esta primera instancia se busca identificar los requerimientos que determinen los atributos de calidad. En el uso tradicional de ATAM son requerimientos específicos del dominio de aplicación del negocio. Para el estudio se utilizan las características de volumen, variedad y velocidad para caracterizar el dominio de big data. Estas son ejemplificadas con escenarios concretos sobre los dominios de aplicación de las arquitecturas de software relevadas.
2. **Descripción de las arquitecturas de software.**
   En esta etapa se describen las arquitecturas candidatas a ser evaluadas, en términos de sus elementos y propiedades, que sean relevantes para el análisis de los atributos de calidad anteriormente identificados como requerimientos del sistema. En el uso tradicional de ATAM se describen alternativas de diseño de una misma arquitectura para evaluar su adecuación a los diferentes requerimientos identificados. Para este

[3] Este estudio utiliza el ATAM presentado originalmente [111] por el SEI que luego es extendido con un mayor detalle de pasos dentro del método [113].

estudio se describen diferentes arquitecturas bajo el dominio general de big data para analizar y comparar sus decisiones de diseño.

3. **Evaluación de las arquitecturas.**

   Se evalúan las arquitecturas planteadas con respecto a los diferentes requerimientos y escenarios de uso. Se analiza como el diseño arquitectónico afecta el comportamiento del sistema para alcanzar los atributos de calidad esperados. En este caso se utilizarán las tácticas [71] como forma de capturar las decisiones de diseño principales de cada arquitectura de software.

4. **Resultados y discusión**

   Una vez identificados los elementos de la arquitectura, se pueden valorar para entender los beneficios y perjuicios sobre cada decisión de diseño tomada. Para este estudio se identificarán las tácticas comunes a las diferentes arquitecturas de software de big data.

4. Mapeo sistemático de arquitecturas de software para big data

En este capítulo se describe el diseño del protocolo y resultados del mapeo sistemático.

4.1. Protocolo del mapeo

4.1.1. Preguntas de investigación

Del objetivo principal se desprenden las siguientes preguntas de investigación que guiarán el estudio de mapeo sistemático.

Q1: ¿Qué propuestas de arquitecturas de software son utilizadas bajo contextos de big data?

Q2: ¿Cómo se pueden clasificar las propuestas de arquitecturas de software de big data?

4.1.2. Proceso de búsqueda

Para iniciar el proceso de búsqueda se definió la cadena de búsqueda a partir de las preguntas de investigación planteadas utilizando el método PICOC (Population, Intervention, Comparison, Outcome, Context) [102] definido en la Tabla 1.

Término	Definición
Population	Propuestas de arquitecturas de software.
Intervention	Fase de diseño arquitectónico en el desarrollo de software.
Comparison	No se realiza una comparación, pero sí se pretende categorizar las propuestas de arquitecturas de software encontradas.
Outcomes	Notaciones, capas y tipos de arquitecturas de software.
Context	Arquitecturas de software teóricas o aplicadas en el contexto de big data.

Tabla 1 Términos del PICOC

En función de las preguntas de investigación y el PICOC aplicado, se realizaron búsquedas exploratorias para evaluar las palabras clave (keywords) e identificar su relevancia dentro de los resultados obtenidos. El objetivo de estas búsquedas fue definir los términos principales y sus sinónimos asociados. La Tabla 2 presenta las palabras clave derivadas a partir de la pregunta de investigación y las búsquedas exploratorias realizadas.

Término 1	Término 2	Término 3
big data	architecture	software
data science	design	

Tabla 2 Términos de búsqueda

A partir de los sinónimos de los distintos términos especificados se genera la cadena de búsqueda. Para ejecutar las búsquedas se utilizó el motor de búsqueda Scopus accesible desde el portal Timbó[4] de la Agencia Nacional de Investigación e Innovación (ANII) de Uruguay. Dentro del buscador se agregaron filtros por área de conocimiento Computer Science. La cadena de búsqueda utilizada fue: (big data OR data science) AND (architecture OR design) AND (software).

4.1.3. Selección de estudios

A continuación se detallan los criterios de inclusión y exclusión del protocolo de búsqueda utilizado para la etapa de filtrado de los artículos.

Se consideran relevantes y se incluyen aquellos resultados de búsqueda que:

- Presenten evidencia de una propuesta de arquitectura de software. Esto descarta propuestas que solo incluyan aspectos de hardware sin elementos de software.
- Las arquitecturas de software deben ser aplicadas en contextos de big data.
- La propuesta debe estar documentada por alguna notación de modelado que facilite la comunicación de los componentes y estructura de la arquitectura de software.
- Estén escritos en inglés.
- Sea posible acceder al texto completo mediante los motores de búsqueda utilizados.

Aquellos estudios que sean relevantes para la investigación y no sean detectados por los motores de búsqueda utilizados serán agregados a la bibliografía en forma manual por parte del investigador.

No se consideran relevantes y se excluyen aquellos resultados de búsquedas que no concuerden con los criterios de inclusión antes mencionados.

[4] http://www.timbo.org.uy

4.1.4. Proceso de clasificación

Petersen sugiere un método de clasificación denominado Keywording [104], el cual permite reducir los tiempos para la definición de un esquema de clasificación y asegurar que la clasificación contempla los artículos procesados en el mapeo. El método consiste en la lectura del resumen para extraer palabras claves y conceptos que reflejen la contribución del artículo. Al avanzar la búsqueda los conjuntos de conceptos son combinados a partir de los diferentes artículos para desarrollar un entendimiento a mayor nivel sobre la naturaleza de la contribución de los resultados de la búsqueda. Esto permite encontrar categorías que son representativas de la población subyacente. Cuando un conjunto final de palabras claves son elegidas, se pueden agrupar y utilizar como forma de categorizar el mapeo.

Este método fue utilizado para contestar la pregunta de investigación Q2 (clasificación de arquitecturas de software) según el siguiente esquema de clasificación.

- **Tipo de arquitectura**: Describe el tipo de arquitectura de software desde el punto de vista de su contexto aplicable.
  - **Industrial**: Describen soluciones de aplicación corporativas de big data dentro del contexto de una organización en particular.
  - **Teóricas**: Describen soluciones más generales en diferentes contextos pero están restringidas a algún problema particular dentro del área de big data.
  - **Referencia**: Describen arquitecturas de referencia que capturan el conocimiento de otras arquitecturas existentes y sirven de guía para los practicantes en sus futuros diseños arquitectónicos.
- **Notación de modelado**: Describe el tipo de notación utilizado para describir la arquitectura de software.
  - **Ninguno**: Estos artículos son filtrados por el criterio de inclusión.
  - **UML**: La especificación de Unified Modeling Language por OMG (Object Management Group) [120]
  - **Otro**: Otra notación propia o diferente de UML.
- **Capa de aplicación**: Describe la capa de big data que modela la arquitectura de software. Dicha clasificación fue generada a partir de las capas de las arquitecturas de referencia encontradas [86], [87], [88], [89].

- **Data storage**: Capa relacionada con el almacenamiento y extracción de datos desde sus diferentes orígenes.
- **Data processing**: Capa relacionada con el preprocesamiento, integración y agregación de los datos para ser posteriormente analizados.
- **Data analysis**: Capa dedicada al procesamiento y análisis de los datos.
- **Data visualization**: Capa que permite la interpretación y visualización de los resultados.

4.2. Resultados

La búsqueda se ejecutó en Scopus en setiembre de 2016 obteniendo como resultado preliminar 792 entradas posibles. Luego de realizar un primer filtro aplicando los criterios de inclusión y exclusión sobre título y resumen, se obtuvieron 207 estudios a ser analizados con mayor detalle. Como resultado de un segundo filtro sobre el texto completo de los artículos, se detectaron 87 artículos que describían arquitecturas de software en contextos de big data. Por otro lado, se incluyeron manualmente 3 estudios que no fueron detectados por la búsqueda de Scopus y que son propuestas de arquitecturas de software relevantes referenciadas como trabajo previo por otras propuestas encontradas. La Tabla 3 presenta los resultados del filtrado.

Fuente	# Artículos	Filtro 1	Filtro 2	Revisión	Total
Scopus	792	207	87	-	87
Otros	-	-	-	3	3

Tabla 3 Filtrado de artículos

A continuación se describen los resultados de los estudios primarios seleccionados[5]. En primer lugar, la Figura 8 presenta la evolución en el tiempo de las propuestas encontradas. Se observa un interés creciente en las propuestas de arquitectura de software en contextos de big data en los últimos años. En 2015 y 2016 se encontraron el 86% del total de las propuestas identificadas en el mapeo. La búsqueda fue ejecutada en setiembre del 2016 por lo que no todos los artículos de ese año fueron recolectados.

[5] El detalle de todos los resultados del mapeo se puede encontrar en la siguiente planilla: https://1drv.ms/x/s!AkWvCfOhluTyhhqIRHAISXe-3z8d

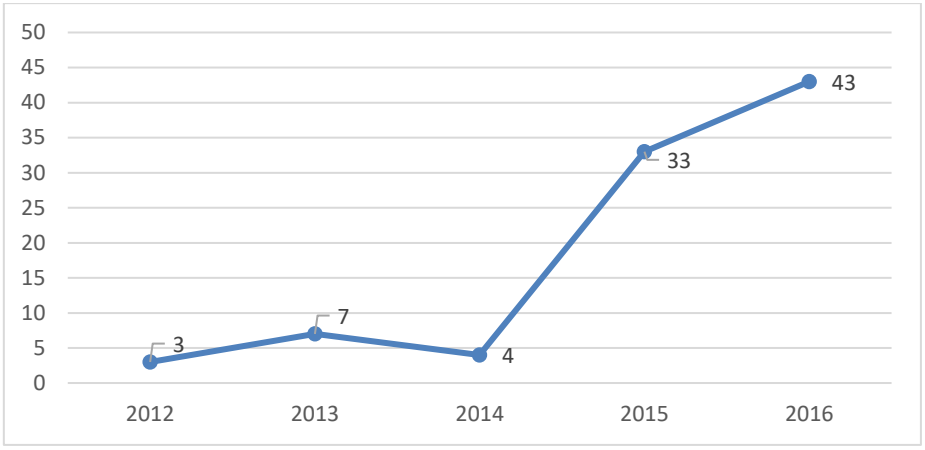

Figura 8 Propuestas de arquitectura de software de big data por año.

A continuación se describen los resultados de la clasificación propuesta para responder la pregunta de investigación Q2. Se encontró que una mayoría de arquitecturas encontradas son teóricas (53%) y de aplicación (38%), frente al 9% de las arquitecturas de referencia, representada en la Figura 9. Solo una minoría de las arquitecturas de software son especificadas utilizando la notación UML (13%).

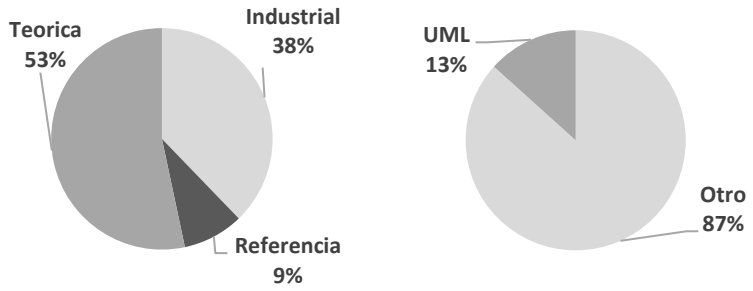

Figura 9 Distribución de tipos de arquitectura y notación utilizada

Es posible decir que las mayorías de arquitecturas de software especifican en múltiples capas (61%), aunque existe un gran número que se centra principalmente en el almacenamiento de datos exclusivamente (21%), en el análisis de datos (10%) y en el procesamiento de datos (8%).

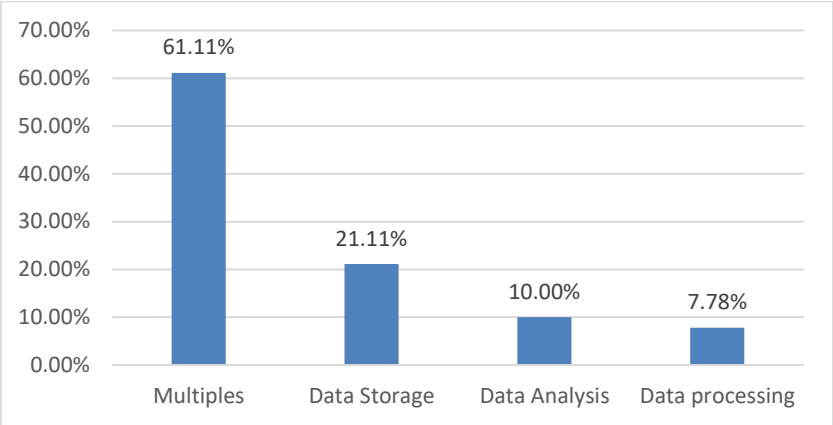

Figura 10 Distribución por capas de arquitectura.

4.3. Amenazas a la validez

Los resultados de un estudio de mapeo pueden ser afectados por una serie de amenazas a la validez [121] que son analizadas a continuación.

4.3.1. Validez interna

La validez interna refiere a la conexión entre lo observado y las explicaciones propuestas [122]. Esto asegura que las conclusiones obtenidas sean verdaderas. Esta amenaza se disminuye a través de un protocolo definido para la búsqueda y selección de artículos con criterios de inclusión y exclusión que permitan delimitar el espacio de investigación.

4.3.2. Validez externa

La validez externa se refiere a la posibilidad de generalización de los resultados [122]. Dado el procedimiento sistemático seguido durante el estudio de mapeo, los resultados y las conclusiones generales derivadas de ellos tienen validez dentro del dominio de investigación abordado, permitiendo que los resultados del estudio puedan ser usados como un punto de partida para otras líneas de investigación.

4.3.3. Validez de conclusión

La validez de conclusión se refiere a la relación entre el tratamiento y el resultado final observado [122]. Una amenaza a la validez de conclusión se puede dar en la extracción de datos. Para disminuir esta amenaza se desarrolló un formulario de extracción de datos que asegura que los datos relevantes extraídos sean consistentes. Las clasificaciones del mapeo se realizaron a

través de la técnica de Keywording [104] y se utilizaron las arquitecturas de referencia como guía para su validación. Existen varias de las clasificaciones más detalladas como dominios de aplicación, frameworks y tecnologías utilizadas en las arquitecturas que se dejan para una posterior investigación y análisis más profundo del mapeo.

4.3.4. Validez del constructo

La validez del constructo asegura que la construcción del estudio esté relacionada con el problema de investigación y que las fuentes seleccionadas sean relevantes [122]. Una posible amenaza a la validez del constructo puede ser el sesgo en el proceso de selección de estudios primarios. Esta amenaza se disminuye con la construcción de un protocolo de investigación. Una amenaza a la validez es que la búsqueda y selección de artículos se realizaron por un único investigador. Para disminuir esta amenaza se utilizaron artículos relevantes previamente seleccionados como muestreo y validación en la inclusión de los resultados del protocolo utilizado.

5. Evaluación de arquitecturas software para big data

El objetivo de esta evaluación es identificar los atributos de calidad comunes en big data y sugerir tácticas de arquitectura que permitan capturar y analizar el impacto sobre las decisiones de diseño en dichos contextos. Para esto, se definirán escenarios de uso generales y atributos de calidad a partir de las características principales de big data. De la evaluación de estos escenarios se podrán identificar y extraer las tácticas comunes para analizar su impacto sobre cada atributo de calidad.

5.1. Recolección de requerimientos y atributos de calidad

5.1.1. Atributos de calidad y escenarios de uso

Los modelos de calidad de software proveen un marco de trabajo para especificar y evaluar requerimientos a través de un conjunto de características o atributos de calidad con sus interrelaciones [123]. Los atributos de calidad [67] son una propiedad medible del producto de software para indicar su grado de satisfacción hacia alguna necesidad de negocio. Existen diferentes modelos de calidad [124] [125] que definen diferentes atributos de calidad para un producto de software. ISO/IEC 25010 [123] estandariza y agrupa 6 características principales con 21 sub-características de calidad comunes en los productos de software.

La habilidad de un sistema para cumplir con sus atributos de calidad es principalmente determinada por su arquitectura. De esto radica la importancia de entender la relación entre la arquitectura de software y los atributos de calidad. Las tácticas son una forma de guiar el diseño de la arquitectura para satisfacer los atributos de calidad que caracterizan el sistema de software a construir. Por otro lado, los escenarios de uso son una forma de describir los requerimientos propios de los atributos de calidad. En la Figura 11 [126] [127] se resumen las relaciones entre escenarios de uso, atributos de calidad y tácticas de arquitectura.

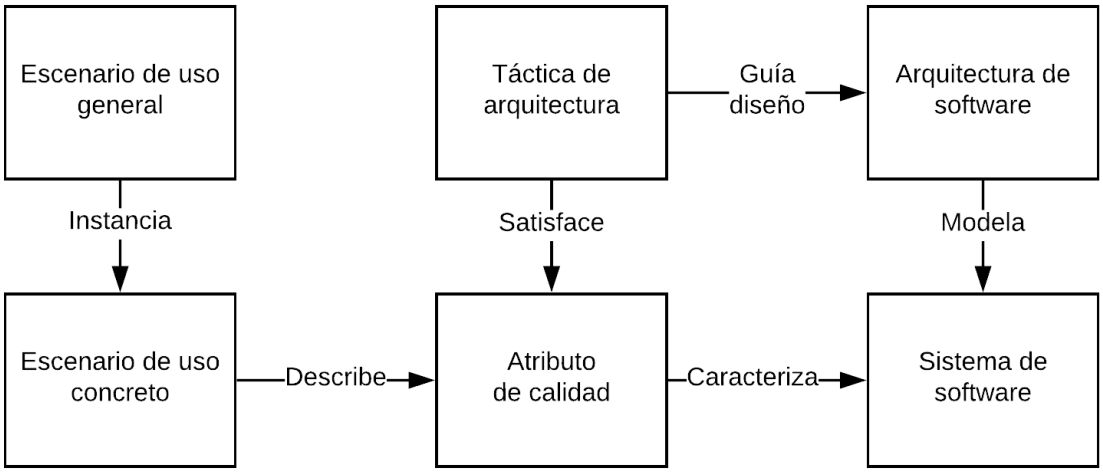

Figura 11 Modelo de escenarios, atributos y tácticas de arquitectura

Dentro de los sistemas de big data se pueden extraer escenarios de uso propios de las características de volumen, velocidad y variedad de datos. Los sistemas de big data deben poder gestionar y procesar grandes volúmenes de datos, con una alta frecuencia (velocidad) en la generación, análisis y publicación de sus resultados, bajo una variedad de formatos (estructurados, semi-estructurados y no estructurados) y fuentes de datos (estáticos y móviles). Para satisfacer estas características de gran volumen, alta velocidad y variedad es necesario distribuir el almacenamiento y procesamiento de los datos bajo múltiples nodos. Esta distribución de datos y procesamiento genera mayores probabilidades de fallas dentro del sistema, que se deben gestionar para su mitigación y recuperación. La Tabla 4 resume las características de big data con sus atributos de calidad, ejemplificados por escenarios de uso concretos de LinkedIn [128] [129] y Twitter [130] [131] [132] [133].

Característica	Atributo de calidad	Escenario de uso general	Ejemplos de escenario de uso concretos
Gran volumen de datos y procesamiento	Escalabilidad	Gestionar grandes volúmenes de datos.	**LinkedIn** Almacenar 100 TB de datos comprimidos en 300 tópicos diferentes. **Twitter** Agregar logs de cientos de TB diarios generados en miles de nodos con información de los usuarios.
		Procesar grandes volúmenes de datos.	**LinkedIn** Procesar 15 billones de mensajes por día. **Twitter** Recolectar estádisticas, categorización y tendencias de 250 millones de tweets diarios
Alta velocidad de datos y procesamiento	Eficiencia	Alta frecuencia de generación y lectura de datos.	**LinkedIn** Procesar picos de escrituras de hasta 200.000 mensajes por segundo. **Twitter** Gestionar un ratio de arribo de 6000 tweets por segundo.
		Alta frecuencia en el análisis y publicación de resultados	**LinkedIn** Procesar un flujo de datos del comportamiento de los usuarios de 1.2 millones eventos por segundo. **Twitter** Procesar tweets para que queden disponibles antes de 10 segundos de ser escritos. 2 billones de consultas con latencia menores a 50ms.
Alta variedad de formatos y estructura datos	Modificabilidad	Múltiples formatos de datos (estructurados, semi-estructurados y no estructurados)	**Twitter** Soportar diferentes formatos de datos como relaciones entre usuarios, tweets y multimedia de imágenes y video. **LinkedIn** Soportar diferentes formatos de datos como relaciones entre usuarios, mensajes, blogs y multimedia de imágenes y video.

Distribución de datos y procesamiento	Disponibilidad	Recuperar ante fallas en el almacenamiento, procesamiento y comunicación de la plataforma.	**Twitter** Recuperar ante caídas en el procesamiento de estadísticas y rankeo de los tweets realizadas en memoria. **LinkedIn** Recuperar ante caídas de los subscriptores en el procesamiento de los diferentes tópicos de la plataforma.
		Tolerar fallas en el almacenamiento, procesamiento y comunicación de la plataforma.	**Twitter** Tolerar caídas en los servicios de consultas y búsquedas de tweets. **LinkedIn** Evitar pérdida de información de los mensajes publicados en los diferentes tópicos de la plataforma.

Tabla 4 Escenarios de uso en big data

5.1.1.1. Escalabilidad

La escalabilidad describe como un sistema responde adecuadamente frente al incremento de carga. Un sistema escalable debería poder ajustar sus recursos para cubrir los cambios de demanda en su uso [67] como se describe en la Tabla 5.

Estímulo	Incremento en la carga (demanda) en un recurso como almacenamiento, procesamiento o comunicación del sistema.
Entorno	Incremento en la carga es transitorio Incremento en la carga es permanente
Respuesta	El sistema provee nuevos recursos para cubrir la demanda
Medida de la respuesta	Tiempo necesario para proveer nuevos recursos ajustados a la nueva demanda Costo de nuevos recursos respecto al valor generado por la nueva demanda

Tabla 5 Escenario de uso general de escalabilidad

Existen dos formas tradicionales de escalar los sistemas: horizontal y vertical. Escalar verticalmente implica incrementar los recursos sobre un único nodo computacional, mientras que escalar horizontalmente implica hacerlo sobre múltiples nodos. El scale cube [134] describe tres dimensiones diferentes de escalabilidad horizontal expresados en tres ejes de descomposición y replicación de los sistemas. El eje X del cubo describe como lograr escalar el procesamiento, a través de la replicación, redundancia y paralelización de los datos, sobre múltiples nodos bajo esquemas de balance de carga. El eje Y aplica una descomposición

funcional del sistema en servicios con responsabilidades similares, como suelen promoverse en arquitecturas de microservicios. El eje Z propone dividir los datos en distintas particiones, lo que usualmente se conoce en base de datos como shards. Todos estos ejes describen diferentes dimensiones que se pueden combinar para escalar horizontalmente los sistemas.

5.1.1.2. Eficiencia

La eficiencia es la capacidad del sistema para responder bajo tiempos específicos. Caracterizar los eventos que suceden dentro del sistema y sus tiempos de respuesta es la esencia principal en la discusión de este atributo de calidad [67] como lo muestra la Tabla 6.

Estímulo	Arribo de eventos o solicitudes esporádicos al sistema Arribo de eventos o solicitudes periódicos al sistema Arribo de eventos o solicitudes estocásticos al sistema
Entorno	Carga intensiva de lecturas/escrituras Carga intensiva de procesamiento Clientes centralizados o distribuidos geográficamente
Respuesta	Se procesa cada solicitud Cambio en el nivel de respuesta del servicio
Medida de la respuesta	Latencia promedio Peor caso de latencia Varianza de latencia Ratio de demanda pico Promedio de demanda

Tabla 6 Escenario de uso general de eficiencia

5.1.1.3. Modificabilidad

La modificabilidad es la facilidad del sistema para analizar, implementar, probar y desplegar modificaciones sin introducir defectos o degradar la calidad del producto actual [123]. Para esto se debe entender el origen, frecuencia, complejidad y costo del cambio como describe la Tabla 7.

Estímulo	Agregar/Eliminar/Variar Funcionalidad Formato/Estructura de datos Capacidad o tecnología de almacenamiento/procesamiento
Entorno	Tiempo de diseño Tiempo de compilación Tiempo de ejecución
Respuesta	Localizar lugares en la arquitectura para realizar cambios Realizar modificaciones sin afectar otras partes del sistema Probar modificaciones Desplegar modificaciones
Medida de la respuesta	Dificultad del cambio en términos de Componentes afectados Esfuerzo/Tiempo/Costo

Tabla 7 Escenario de uso general de modificabilidad

5.1.1.4. Disponibilidad

La disponibilidad refiere a la habilidad del sistema para enmascarar o reparar faltas que puedan provocar fallas e inconvenientes a los usuarios finales [67]. En sistemas distribuidos, la probabilidad de fallas en el sistema se incrementa, con el aumento en la multiplicidad e interconexión de sus nodos. Los sistemas deben manejar estos errores para mitigar las fallas que puedan suceder en tiempo de ejecución como lo muestra la Tabla 8.

Estímulo	Una falla sucede en un recurso del sistema. El recurso puede ser un nodo de procesamiento, comunicación o almacenamiento.
Entorno	Ocurren una o multiples fallas simultaneamente
Respuesta	El sistema continua procesando solicitudes sin degradar su nivel de servicio. El sistema procesa solicitudes pero degradando su nivel de servicio.
Medida de la respuesta	Tiempo para restaurar el recurso ante una falla temporaria Tiempo para sustituir el recurso ante una falla permanente Cantidad de fallas que pueden ser enmascaradas o toleradas.

Tabla 8 Escenario de uso general de disponibilidad

5.2. Descripción de arquitecturas de software de big data

Para la descripción de las arquitecturas de software se quiere obtener un muestreo representativo de los diferentes tipos de arquitecturas de big data relevadas durante el mapeo sistemático. Con este muestreo se intenta cubrir casos de arquitecturas industriales (**LinkedIn**,

Twitter) en funcionamiento y documentadas, casos teóricos (**Lambda**, **Kappa**, **SOLID**) que son ampliamente discutidos en la comunidad de arquitectos de software, así como las arquitecturas de referencia (**Maier**, **Pääkkönen, Geerdink**, **Viana)** principales.

5.2.1. Arquitecturas industriales

Existen varias empresas que necesitan resolver sus problemáticas de big data bajo sus contextos organizacionales. A continuación se describen las arquitecturas de software de las organizaciones encontradas en la literatura.

5.2.1.1. Arquitecturas de big data en LinkedIn

LinkedIn analiza los datos generados por sus sitios y aplicaciones para derivar la visión e ideas de sus nuevas funcionalidades. El sistema debe manejar alrededor de 100 TB de datos comprimidos en aproximadamente 300 tópicos diferentes. Dicha infraestructura debe procesar unos 15 billones de mensajes por día con picos sostenidos de hasta 200.000 mensajes por segundo. Su arquitectura [128] utiliza un conjunto de sistemas de mensajería distribuidos [129] con procesamiento en lotes [49] coordinados por un planificador de tareas. La Figura 12 muestra la arquitectura de LinkedIn utilizada por sus analistas de datos para procesar la información de sus sistemas.

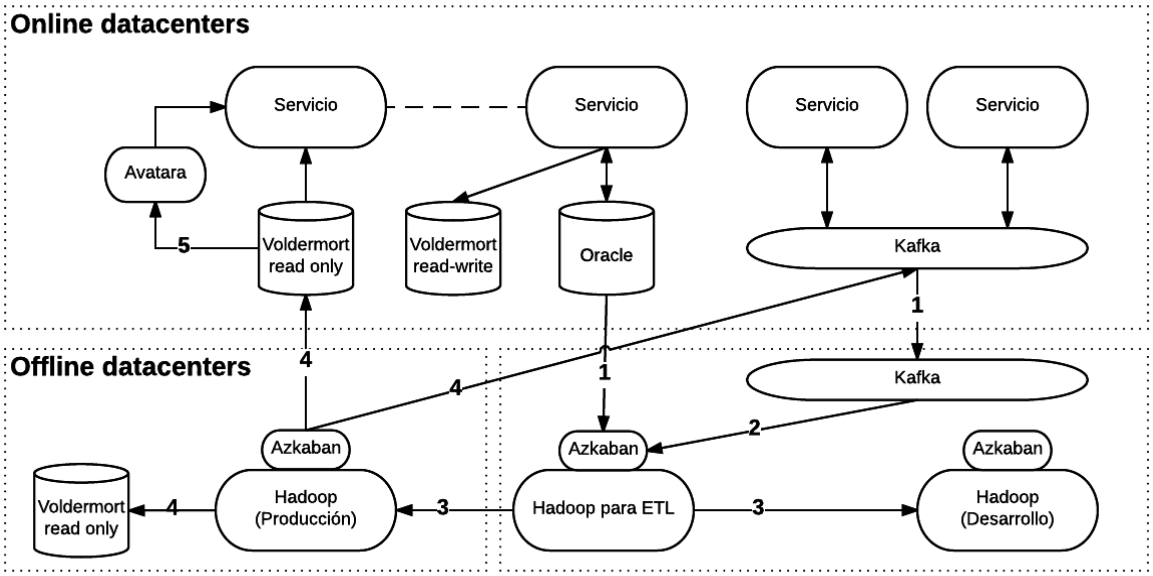

Figura 12 Arquitectura de LinkedIn

1. Los datos son recolectados de dos orígenes principales:
    a. Snapshots de la base de datos sobre información de los usuarios, compañías, conexiones y otros datos primarios del sitio.
    b. Las actividades de los usuarios que se generan como un flujo de eventos basados en el uso de los servicios de LinkedIn.
2. Kafka [129] es un sistema de mensajería (productor-consumidor) distribuido que es usado para recolectar el flujo de eventos de los usuarios. Los productores reportan los eventos agrupados por tópicos a un broker. De esta forma los consumidores pueden leer los eventos a su propio ritmo. La información de los eventos son transferidos a un cluster de Hadoop [49] ETL para su posterior procesamiento.
3. Los datos en el cluster de Hadoop ETL son replicados a clusters de producción y desarrollo. Azkaban es usado como planificador de trabajo soportando una diversidad de tipos de trabajos (MapReduce, Pig, Shell scripts y Hive). Típicamente los trabajos son probados en el cluster de desarrollo antes de ser puestos en producción.
4. Los resultados de los análisis en el ambiente de producción son transferidos a una base de datos para facilitar su depuración frente a posibles incidentes. Los resultados pueden ser devueltos al cluster de Kafka para ser consumidos por los servicios de la aplicación.
5. Avatara [135] es utilizado para la preparación de datos de OLAP. Los datos analizados son leídos desde una base de datos Voldemort para su preprocesamiento, agregación y cubificación de OLAP para finalmente ser persistidos en una nueva base de datos Voldermort.

Para manejar las conexiones de su red social se utiliza un servicio de miembros de la plataforma basado en grafos distribuidos [136]. Para el contenido del sitio se utiliza una base relacional de Oracle, que fue migrada posteriormente a una fuente única de datos documental bajo Espresso [137].

5.2.1.2. Arquitecturas de big data en Twitter

Twitter es una plataforma de comunicación donde sus usuarios pueden escribir mensajes de 280 caracteres de largo llamados “tweets” a sus seguidores (otros usuarios que se subscriben a dichos mensajes). Twitter tiene más de 100 millones de usuarios activos mundialmente, que colectivamente envían más de 250 millones de tweets diarios. Para la búsqueda de tweets la aplicación cuenta con un buscador que debe servir más de 2 billones de consultas diarias con

una latencia promedio de 50 ms. Su arquitectura [130] fue diseñada para que los tweets puedan ser encontrados luego de 10 segundos de ser creados con procesamientos en tiempo real. Existe el desafío adicional de guardar los patrones de uso [138] de los usuarios para su posterior análisis [139] bajo procesamiento por lotes. La Figura 13 muestra la arquitectura del buscador de tiempo real de Twitter [130]. La arquitectura original estaba basada en el stack de Hadoop para su procesamiento en lotes. Sin embargo, los requerimientos de tiempo real de la búsqueda exigían tiempos de latencia que no eran alcanzados por la configuración original.

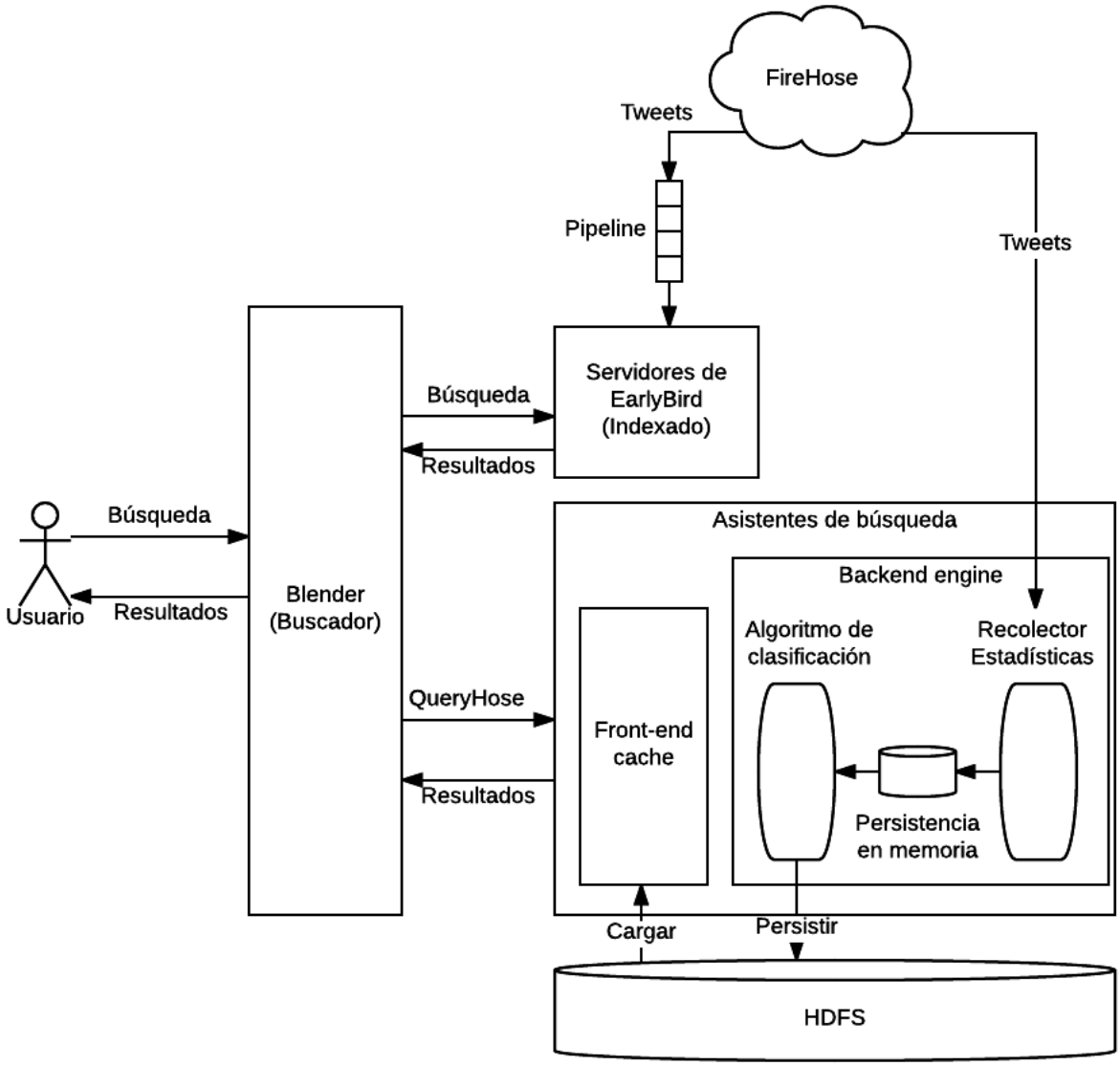

Figura 13 Arquitectura del buscador de tiempo real de Twitter

En la nueva arquitectura de Twitter existe un front-end denominado Blender que recibe todas las búsquedas de la plataforma. Las consultas incluyen la búsqueda de tweets y cuentas de usuario a través de un servicio denominado QueryHose. Los tweets son creados por otro

servicio (“FireHose”) dirigidos a un pipeline para su anotación y conversión. Luego dichos tweets son enviados a los servidores de EarlyBird para su filtrado, personalización e indexado. EarlyBird [132] es un motor de búsqueda de tiempo real diseñado para proveer consultas con baja latencia y alto rendimiento. La arquitectura cuenta con asistentes de búsqueda adicionales como recolectores de estadísticas y algoritmos para la clasificación de los tweets. Estos análisis son persistidos en el sistema de HDFS de Hadoop para su posterior uso por los servicios de la plataforma.

Existen otras arquitecturas industriales en la literatura como **Facebook** [140] y **Netflix** [141] que demuestran los desafíos de escalabilidad, disponibilidad y performance de los sistemas de big data [79].

5.2.2. Arquitecturas teóricas

En este apartado se describen las arquitecturas teóricas de big data encontradas en la literatura que proponen soluciones más generales aplicables a un mayor número de contextos.

5.2.2.1. Arquitectura lambda

La arquitectura **Lambda** [142] es una arquitectura de procesamiento de datos, diseñada para gestionar un volumen masivo de datos, combinando el procesamiento por lotes (batch-processing) con el procesamiento de flujo de datos en tiempo real (stream-processing) [143]. Esta aproximación permite balancear la latencia, escalabilidad y disponibilidad necesarias para una solución de big data. La escalabilidad y precisión de las consultas se desarrolla con el procesamiento por lotes, mientras que la baja latencia y disponibilidad se construye con el procesamiento de datos en tiempo real. Esto permite alcanzar la consistencia eventual [35] mientras el desfasaje de la capa por lotes (batch layer) se sincroniza con la capa de tiempo real (speed layer). La Figura 14 muestra las capas que componen la arquitectura lambda. Las principales propiedades y relaciones entre capas se resumen a continuación [144].

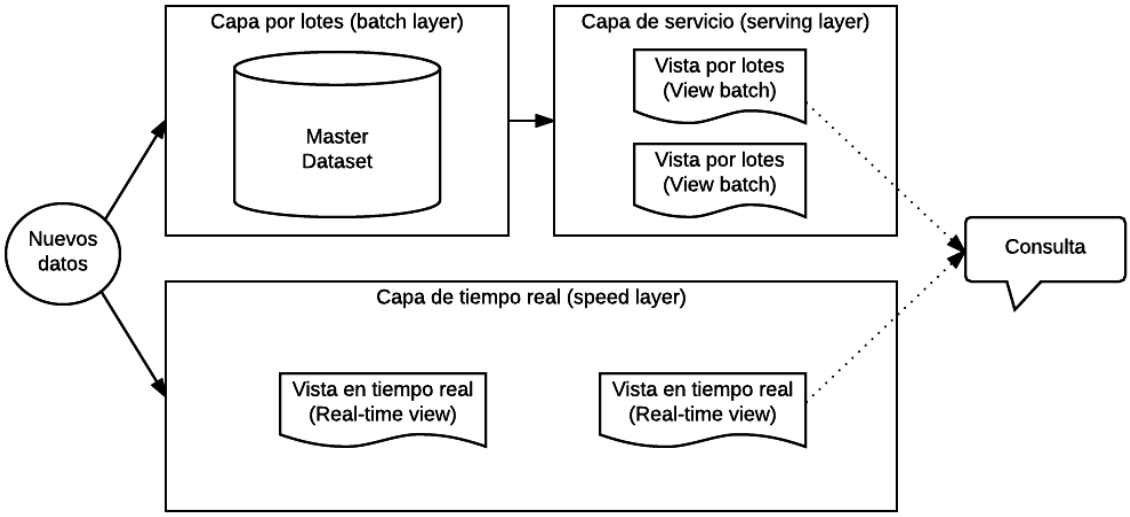

Figura 14 Arquitectura Lambda

1. Todos los datos que ingresan al sistema son despachados a la capa por lotes (batch layer) y a la capa de flujo en tiempo real (speed layer) simultáneamente para su procesamiento.
2. La capa de lotes (batch layer) tiene dos funciones principales:
    a. Gestionar el conjunto de datos maestros (master dataset). Es un conjunto de datos inmutables representando toda la información del sistema. Una vez almacenados no pueden ser alterados.
    b. Gestionar el preprocesamiento y cálculo de las vistas por lotes (batch views).
3. La capa de servicio (serving layer) indexa las vistas por lotes (batch views) para que puedan ser consultadas con baja latencia y alta velocidad de acceso.
4. La capa de flujo en tiempo real (speed layer) compensa la alta latencia de actualizaciones y sincronización de la capa de servicio (serving layer) procesando únicamente los datos recientes del sistema.
5. Toda consulta del sistema puede ser resuelta combinando los resultados de las vistas de lotes (batch views) y flujo en tiempo real (real-time views).

5.2.2.2. Arquitectura kappa

La arquitectura **Kappa** [145] es una simplificación de la arquitectura Lambda. Utiliza un log inmutable pero remueve la complejidad de la capa de procesamiento por lotes. Para lograr esta simplificación utiliza una única capa de procesamiento de tiempo real. Sin embargo, está capa puede ir evolucionando con nuevas lógicas de procesamiento necesarias para servir nuevos resultados. Para esto se utiliza el log como almacenamiento canónico de los datos originales y

se promueven nuevas versiones de procesamiento de los datos. Se utiliza un sistema de versionado para ir liberando las nuevas actualizaciones del procesamiento en producción como se muestra en la Figura 15.

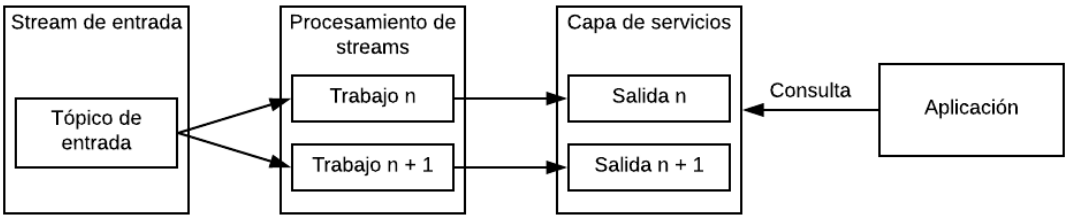

Figura 15 Arquitectura Kappa

5.2.2.3. Arquitectura SOLID

La arquitectura SOLID (Service-OnLine-Index-Data) [146] surge para resolver problemas de big data en sistemas de tiempo real. La arquitectura se compone de capas que separan la complejidad de gestionar grandes volúmenes de datos con respecto a su generación y consumo en tiempo real.

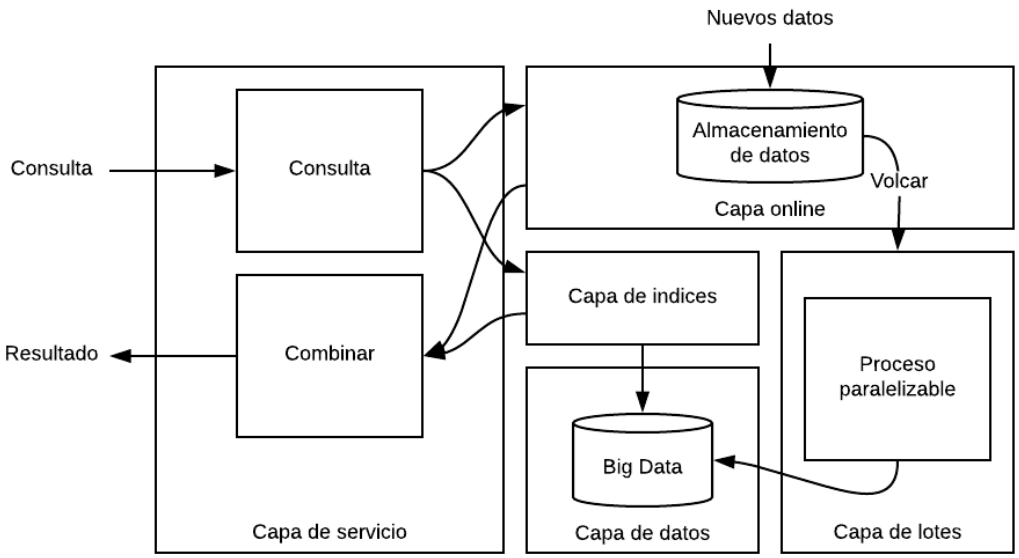

Figura 16 Arquitectura SOLID

La Figura 16 muestra las diferentes capas de la arquitectura que se resumen a continuación.

- **Capa de tiempo real (Online Layer)**
  La capa superior resuelve las necesidades de acceso en tiempo real de los datos de entrada. Los nuevos datos son capturados en almacenamientos temporarios de alta velocidad de acceso. Estos repositorios son utilizados como un gran buffer para generar y consumir los datos recientes de la aplicación (todos aquellos datos que no existen todavía en el almacenamiento principal).
- **Capa de datos (Data Layer)**
  La capa de datos contiene el almacenamiento principal diseñado para almacenar grandes volúmenes de datos, su organización y semántica. Este almacenamiento puede tener alta latencia y tiempos de respuesta lentos.
- **Capa de índices (Index Layer)**
  Esta capa provee los índices para la capa de datos (Data Layer) con una alta disponibilidad y velocidad de acceso. Los índices son generados con la semántica de los datos de la capa inferior con el objetivo de proveer un rápido acceso para las consultas necesarias de la aplicación.
- **Capa de servicios (Service Layer)**
  La capa de servicios se presenta como una fachada para el usuario externo de la aplicación. Todas las consultas son realizadas a través de esta capa que redirige las consultas a las capas de tiempo real e índices para resolverlas y combinarlas para conseguir la respuesta final.
- **Capa de combinación (Merge Layer)**
  La capa de combinación recibe y transforma los datos de entrada para combinarlos con los datos existentes en el almacenamiento principal. Esto implica generalmente un procesamiento intenso de datos por lo que debe ser procesado en lotes usando alguna técnica de paralelización como puede ser MapReduce [54].

5.2.3. Arquitecturas de referencia

Durante los años recientes surgen arquitecturas de referencia [86] [87] [88] [89] que tratan de abstraer y capturar el conocimiento de las arquitecturas teóricas y concretas de la industria. Angelov [147] clasifica las arquitecturas de referencia en 5 tipos resumidos en la Tabla 9.

Tipo	1	2	3	4	5
Porque	Estandarización	Estandarización	Facilitación	Facilitación	Facilitación
Cuando	Múltiples organizaciones	Única organización	Múltiples organizaciones	Única organización	Múltiples organizaciones
Quién	Organizaciones de estandarización, usuarios o arquitectura	Grupo de usuarios o arquitectos	Organización independiente de software o usuarios	Grupo de usuarios o arquitectos	Centros de investigación, organizaciones de arquitectura o usuarios
Cuando	Clásica	Clásica	Clásica	Clásica	Preliminar
Qué	Componentes, interfaces y políticas	Componentes, interfaces y políticas	Componentes, interfaces y políticas	Componentes y políticas	Componentes, algoritmos y protocolos
Como	Semi formal	Formal o semi formal	Semi formal	Semi formal o informal	Formal o semi formal

Tabla 9 Tipos de arquitectura de referencia

La totalidad de las arquitecturas de referencia encontradas y detalladas a continuación se clasifican dentro del tipo 3.

5.2.3.1. Arquitecturas de referencia de tipo 3

Las arquitecturas de referencia de tipo 3 contienen las siguientes principales características:

- El objetivo principal es facilitar y guiar a los arquitectos en los diseños de nuevas soluciones. Esto demuestra que todavía el área no tiene la madurez suficiente para lograr el objetivo de estandarización.
- El contexto es lo suficientemente genérico para aplicarlo en múltiples organizaciones en búsqueda de los principios de diseño arquitectónico. La audiencia principal son los arquitectos con la responsabilidad suficiente para tomar las decisiones de implementación de soluciones de big data dentro de su organización.
- Las arquitecturas se basan en soluciones existentes utilizadas y probadas por la industria.
- Los componentes, interfaces y políticas de la arquitectura se especifican a través de algún lenguaje semi-formal como diagramas.

Maier [86] en su tesis de maestría describe una arquitectura de referencia con el propósito de facilitar y guiar la creación de soluciones de big data para múltiples organizaciones. La arquitectura (Figura 17) se especifica a través de diagramas de UML bajo la arquitectura de vistas 4+1 de Krutchen [64]. A continuación se resumen las principales capas de la aplicación.

1. **Adquisición de datos**: Los datos pueden ser extraídos de un origen de datos dinámico, en el caso de un flujo de datos, o estático con datos en reposo. La extracción puede incluir la necesidad de filtrar los datos, que pueden ser estructurados, semi-estructurados o no estructurados.
2. **Extracción de información**: Para lograr generar información es necesario estructurar los datos con metadatos a través del reconocimiento y clasificación de entidades, así como sus relaciones. Se pueden utilizar ontologías para facilitar el entendimiento y extracción de los modelos propios de los datos.
3. **Gestión de calidad de datos**: La limpieza de datos permite la corrección de errores u omisión en los datos como pueden ser atributos faltantes, inconsistentes o duplicados.
4. **Integración de datos**: La integración permite armonizar los datos de diferentes orígenes a través de un esquema común para unificar las posibles consultas.
5. **Análisis de datos**: El análisis permite derivar significado e interpretación de los datos a través de la agregación de la información sobre diferentes dimensiones.
6. **Distribución de datos**: El objetivo de la distribución de datos es hacer disponible los resultados del análisis a través de interfaces de usuario o aplicación para los usuarios finales.

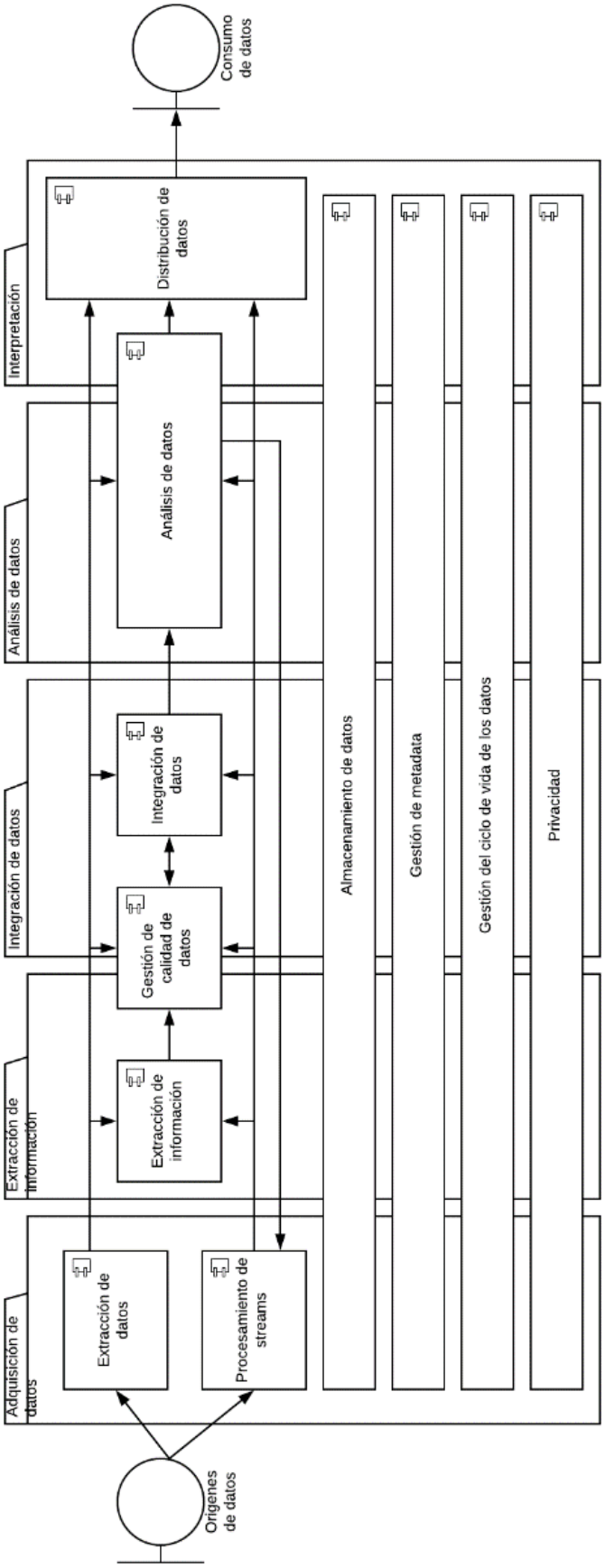

Figura 17 Arquitectura de referencia de Maier

A su vez **Pääkkönen** [88] propone una arquitectura de referencia con el objetivo de facilitar el diseño y elección de tecnologías para su implementación. En la Figura 18 se muestra el diseño de alto nivel de dicha arquitectura que se resume a continuación.

El origen de datos se define en dos dimensiones: movilidad y estructura. La movilidad refiere al dinamismo y velocidad de los datos, mientras que la estructura define la información y metadata disponible para modelar los datos. La extracción puede incluir la transferencia, carga y compresión de los datos como forma de preprocesamiento. Los datos pueden ser combinados y limpiados para mejorar su calidad, así como replicados para su distribución. El proceso de extracción de información logra incorporar una mayor estructura a los datos más desestructurados. Los datos pueden ser analizados con procesamiento por lotes o en tiempo real y transformados para ser visualizados. La visualización puede ser realizada a través de dashboards o herramientas de reportes para el usuario final.

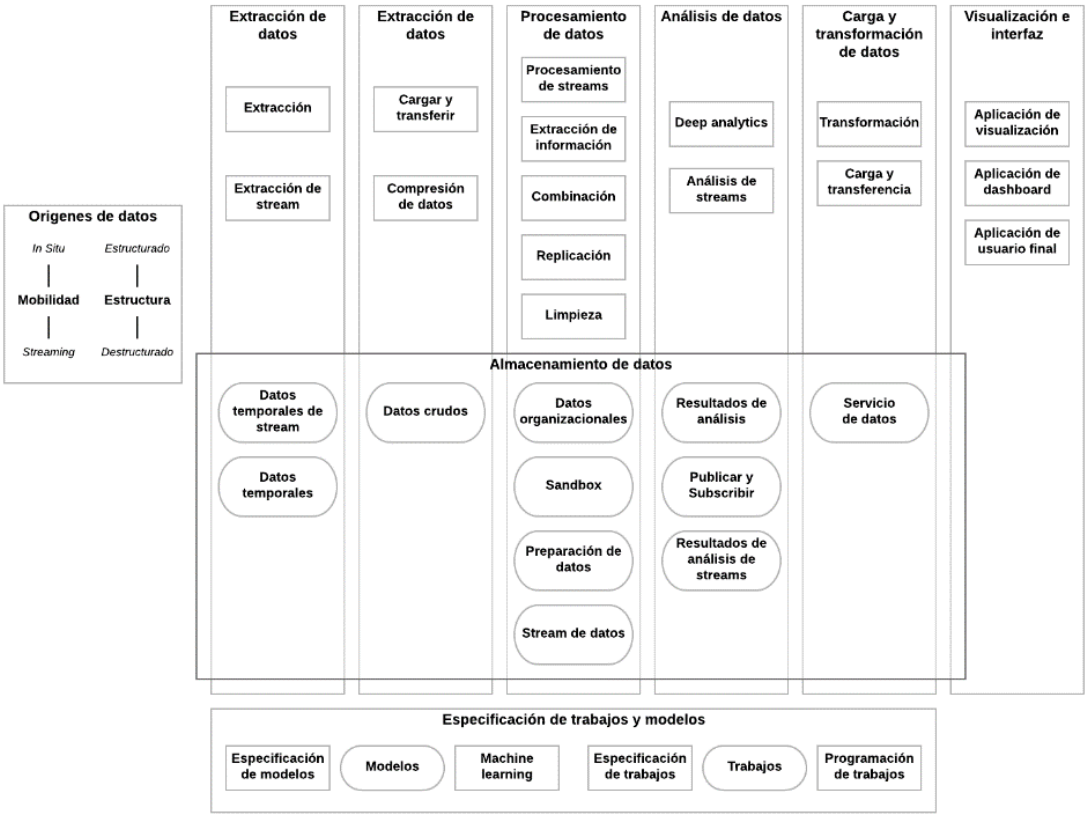

Figura 18 Arquitectura de referencia de Pääkkönen

Geerdink [87] detalla una arquitectura de referencia basada en dos patrones principales: layers y pipes & filters [73]. Los artefactos de la arquitectura (Figura 19) son creados iterando sobre el paradigma de diseño planteado por Hevner [148]. La arquitectura se compone de tres

capas: negocio, aplicación y tecnológica. Se puede apreciar el uso de pipes & filters en la capa de negocio con la secuencia de importación, procesamiento, análisis y decisión de los datos.

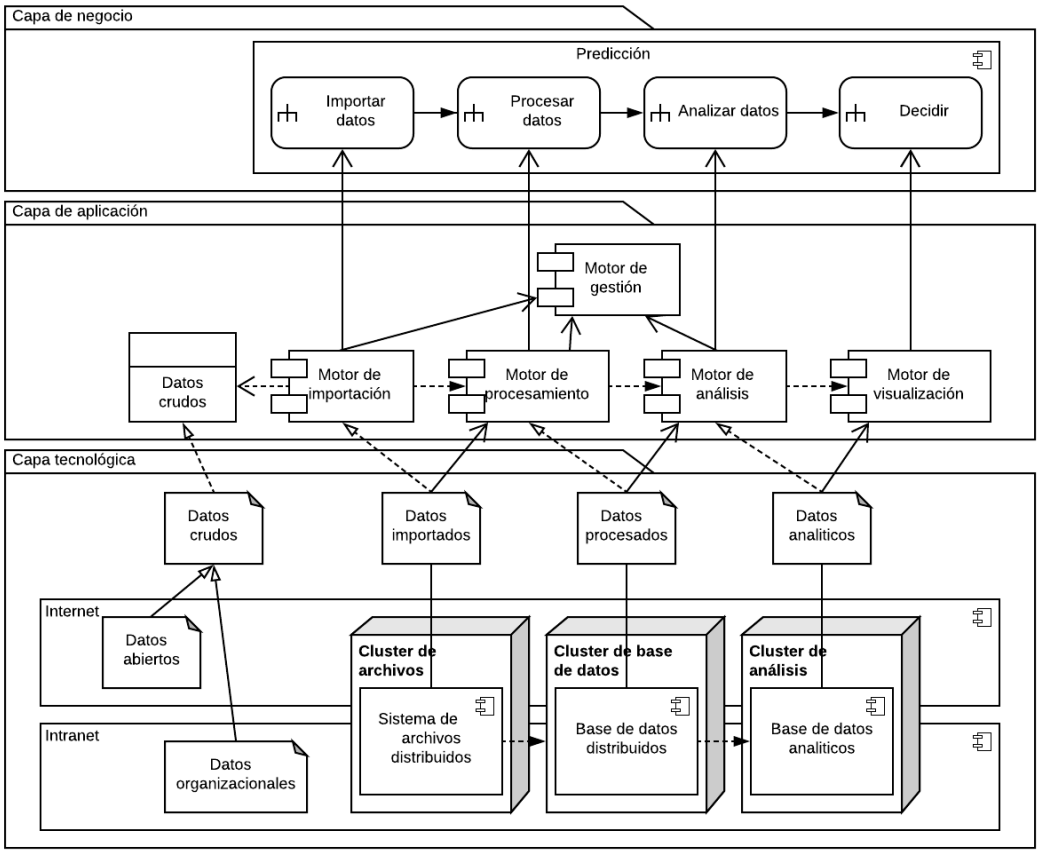

Figura 19 Arquitectura de referencia de Geerdink

Viana [89] focaliza su estudio en una arquitectura de referencia (Figura 20) para archivar, preservar y recuperar big data en el largo plazo. Existen otros estudios [149][150][151] sobre algunas capacidades puntuales de archivado, preservación y recuperación para datos estructurados y no estructurados.

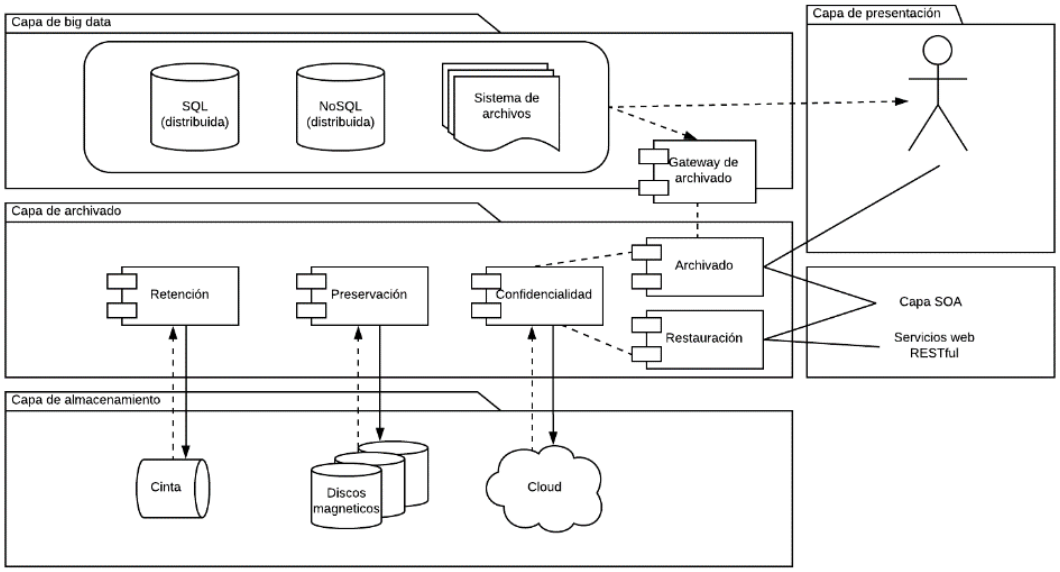

Figura 20 Arquitectura de referencia de Viana

5.3. Evaluación de arquitecturas de software

Para la evaluación de las arquitecturas de software se eligen las arquitecturas con mayor descripción y análisis disponible sobre sus decisiones de diseño para poder demostrar la aplicación de las tácticas de arquitecturas. La selección cubre los diferentes tipos de arquitectura con dos arquitecturas corporativas (**LinkedIn**, **Twitter**), dos teóricas (**Lambda**, **Kappa**) y una de referencia (**Pääkkönen**) previamente descriptas. Las tácticas de arquitectura (mostradas en cursiva) representan las decisiones de diseño más importantes utilizadas en cada arquitectura como resume la Tabla 10.

Arquitectura	Escenarios de uso	Tácticas principales
LinkedIn	15 billones de mensajes por día con picos de hasta 200.000 mensajes por segundo. 100 TB de datos comprimidos en 300 tópicos diferentes.	*Write Ahead Logs* *Procesamiento por lotes* *Procesamiento en tiempo real* *Consistencia eventual* *At least once call semantics* *Entornos shared-nothing* *Persistencia poliglota*
Twitter	100 millones de usuarios activos. 250 millones de tweets diarios. 2 billones de consultas diarias con latencia promedio de 50ms. Tweets visibles antes de 10 segundos después de ser creados.	*Write Ahead Logs* *Procesamiento por lotes* *Procesamiento en tiempo real* *Semánticas de llamadas remotas* *Entornos shared-nothing* *Persistencia poliglota*
Lambda	Linealmente escalable horizontalmente. Tolerante a fallas de hardware. Baja latencia de grandes volúmenes de escrituras/lecturas y su procesamiento asociado.	*Write Ahead Logs* *Procesamiento por lotes* *Procesamiento en tiempo real* *Consistencia eventual*
Kappa	Similar a lambda pero simplificando el modelo computacional para su implementación.	*Write Ahead Logs* *Procesamiento en tiempo real* *Consistencia eventual*
Arquitectura de Referencia de Pääkkönen	Arquitectura de referencia de tipo 3 para guiar el diseño de arquitecturas de software de big data.	*Write Ahead Logs* *Procesamiento por lotes* *Procesamiento en tiempo real* *Replicación de datos* *Entornos shared-nothing*

Tabla 10 Tácticas de arquitecturas de big data

LinkedIn utiliza Kafka [129] como un sistema de mensajería (productor-consumidor) distribuido que es utilizado para recolectar el flujo de eventos de los usuarios bajo un sistema de *write-ahead logs*. Los productores reportan los eventos agrupados por tópicos a un broker [152]. Estos tópicos son *replicados* en diferentes brokers para asegurar su disponibilidad ante pérdidas de información. De esta forma, se almacenan 100TB de datos comprimidos en 300 tópicos diferentes que los consumidores pueden leer a su propio ritmo. La información de los eventos son transferidos a un cluster de Hadoop [91] para su posterior *procesamiento por lotes* con Map-Reduce [54] para procesar 15 billones de mensajes por día. Por otro lado, se utiliza Samza para *procesar en tiempo real* varios de los tópicos. Los datos son persistidos bajo un sistema HDFS (Hadoop Distributed File System) [49] de *entorno share-nothing* y son sincronizados por un sistema de comunicación de datos bajo una *semántica de llamada "at least once"* denominado Gobblin. Toda la arquitectura permite tener una alta disponibilidad

bajo *consistencia eventual* [153], logrando procesar picos de escrituras de hasta 200.000 mensajes por segundo.

LinkedIn ha evolucionado desde un servicio monolito (denominado Leo) [154], con una única base de datos relacional (Oracle), a más de 750 microservicios utilizando diferentes esquemas de *persistencia poliglota* como son grafos (GraphDB), clave-valor (Voldemort[6]), documental (Espresso) [155] y un almacenamiento de blobs (Ambry) [156]. La Tabla 11 resume los atributos de calidad principales asociados con los escenarios de uso y tácticas de arquitectura para LinkedIn.

Atributo de calidad	Escenario de uso concreto	Tácticas principales
Escalabilidad	Almacenar 100 TB de datos comprimidos en 300 tópicos diferentes.	*Particionamiento de datos* con sistema de mensajería distribuido manejando múltiples tópicos.
Escalabilidad	Procesar 15 billones de mensajes por día.	*Procesamiento por lotes* con Map Reduce con sincronización de flujos de trabajos.
Eficiencia	Procesar picos de escrituras de hasta 200.000 mensajes por segundo.	*Consistencia eventual* a través de un bus de datos que sincroniza cambios de diferentes base de datos.
Eficiencia	Procesar flujo de datos de comportamiento de usuarios de 1.2 millones eventos por segundo.	*Procesamiento en tiempo real* con stream de datos.
Disponibilidad	Recuperación ante caídas de los subscriptores en el procesamiento de los diferentes tópicos de la plataforma.	*Write ahead logs* inmutables con integradores de datos que utilizan *semánticas de llamadas remotas* para asegurar los resultados de remotar el procesamiento distribuido.
Disponibilidad	Evitar pérdida de información de los mensajes en los diferentes tópicos.	*Replicación basada en líder* permite tener varias copias de las particiones de los tópicos.
Modificabilidad	Soportar diferentes formatos de datos como relaciones entre usuarios, mensajes, blogs y multimedia de imágenes y video.	*Persistencia poliglota* para diferentes formatos de datos utilizando base de datos de grafos, clave-valor y blobs de almacenamiento.

Tabla 11 Tácticas principales para LinkedIn

Twitter, por su lado, utiliza un front-end denominado Blender [130] que recibe todas las búsquedas de la plataforma. Las consultas incluyen la búsqueda de tweets y cuentas de usuario a través de un servicio denominado QueryHose. Los tweets son creados por otro servicio ("FireHose") dirigidos a un pipeline para su anotación y conversión. Luego dichos tweets son enviados a los servidores de EarlyBird para su filtrado, personalización e indexado, debiendo

[6] https://www.project-voldemort.com/voldemort/

quedar disponibles 10 segundos después de ser creados. EarlyBird [132] es un motor de búsqueda y *procesamiento en tiempo real* diseñado para proveer 2 billones de consultas con latencia menores a 50ms. Para la consistencia de datos se utilizan índices que generan prefijos y aseguran el ordenamiento temporal de los tweets. La arquitectura cuenta con asistentes de búsqueda adicionales, como recolectores de estadísticas y algoritmos, para la clasificación de 250 millones tweets diarios, utilizando algoritmos de *procesamiento por lotes* con Map Reduce. Estos análisis de cientos de TB diarios son persistidos y combinados con *write-ahead logs* [131] en HDFS para su posterior uso por los servicios de la plataforma.

La comunicación con el front-end se realiza con servicios de Thrift [157] y Protocol Buffer [158] utilizando diferentes *semánticas de llamadas remotas* para la recuperación de mensajes ante caídas del sistema. Twitter originalmente fue construido sobre MySQL [159] (incluso teniendo su propio fork[7] de desarrollo). En abril de 2010 se construye Gizzard [160] para mover datos entre las diferentes instancias distribuidas de MySQL. Se utiliza FlockDB [161] como almacenamiento de grafos de las relaciones y conexiones entre los usuarios. En ese mismo año se utiliza Hadoop (originalmente con la intención de realizar respaldos de MySQL) pero posteriormente utilizado para realizar análisis de datos. En el 2012 se utiliza Blobstore [162] para almacenar datos no estructurados como imágenes y videos. En 2014 se incorpora Manhattan [163], una base de datos clave-valor distribuida con consistencia eventual. La Tabla 12 resume los atributos de calidad principales asociados con los escenarios de uso y tácticas de arquitectura para Twitter.

[7] https://github.com/twitter/mysql

Atributo de calidad	Escenario de uso concreto	Tácticas principales
Escalabilidad	Agregar logs de cientos de TB diarios generados en miles de nodos con información de los usuarios.	*Particionamiento de datos* con sistema de logs distribuidos.
Escalabilidad	Recolectar estádisticas, categorización y tendencias de 250 millones de tweets diarios.	*Procesamiento en lotes* con Map Reduce con creación de trabajos y planificación/sincronización del procesamiento.
Eficiencia	Gestionar un ratio de arribo de 6000 tweets por segundo.	*Consistencia por sesión* con la utilización de prefijos para manejar la concurrencia de escrituras/lecturas.
Eficiencia	Procesar tweets para que queden disponibles antes de 10 segundos de ser escritos. 2 billones de consultas con latencia menores a 50ms.	*Procesamiento en tiempo real* con stream de datos con motores de búsqueda e indexado.
Disponibilidad	Recuperar ante caídas en el procesamiento de estadísticas y rankeo de los tweets realizados en memoria.	*Write ahead logs* que son serializados en HDFS cada 5 minutos y agregados con diferentes protocolos de *semánticas de llamadas remotas* bajo *entornos-shared nothing* sobre múltiples nodos independientes.
Disponibilidad	Evitar caídas en los servicios de consultas y búsquedas de tweets.	*Replicación del frontend* con caching sincronizado cada minuto para servir las consultas y búsquedas sobre los tweets.
Modificabilidad	Soportar diferentes formatos de datos como relaciones entre usuarios, tweets y multimedia de imágenes y video.	*Persistencia poliglota* para diferentes formatos de datos utilizando base de datos de grafos, relacional, clave-valor y blobs de almacenamiento.

Tabla 12 Tácticas principales para Twitter

En la arquitectura **Lambda** los datos son almacenados en *write-ahead logs* inmutables que son despachados a la capa por lotes y tiempo real. La capa de *procesamiento por lotes* tiene la responsabilidad de pre-procesar vistas generalmente con algoritmos de Map Reduce. La capa de *procesamiento en tiempo real* procesa el flujo de los últimos datos ingeridos por el sistema. Todas las consultas se combinan entre las vistas por lotes con los datos procesados en tiempo real en una capa de servicios de baja latencia. Esta es una arquitectura que muestra cómo se pueden combinar los procesamientos por lotes y tiempo real.

Atributo de calidad	Escenario de uso general	Tácticas principales
Escalabilidad	Procesar grandes volumenes de datos.	*Procesamiento en lotes* con una capa de servicio de vistas materializadas.
Eficiencia	Alta frecuencia en el análisis y publicación de resultados.	*Procesamiento en tiempo real* con stream de datos.
Disponibilidad	Recuperar ante caídas en el procesamiento distribuido de datos.	*Write ahead logs* como única fuente de datos que sirve a las capas de lotes y tiempo real.

Tabla 13 Tácticas principales para Lambda

La arquitectura **Kappa** demuestra cómo se puede simplificar los procesamientos por lotes de la arquitectura Lambda con versiones completas de *procesamiento en tiempo real*. El versionado de las capas de tiempo real permite evolucionar los cambios en la lógica de procesamiento en forma incremental evitando la necesidad de procesamiento por lotes de mayor latencia.

Atributo de calidad	Escenario de uso general	Tácticas principales
Escalabilidad	Procesar grandes volumenes de datos	*Procesamiento en tiempo real* con versionado de capas para desplegar incrementos de cambios en la lógica de procesamiento.
Eficiencia	Alta frecuencia en el análisis y publicación de resultados	*Procesamiento en tiempo real* con stream de datos versionados.
Disponibilidad	Recuperar ante caídas en el procesamiento distribuido de datos.	*Write ahead logs* como única fuente de datos que sirve a las distintas versiones de las capas de tiempo real.

Tabla 14 Tácticas principales para Kappa

Pääkkönen propone una arquitectura de referencia en varias capas. La capa de datos clasifica la movilidad de datos bajo dos formatos (in situ y streams). Los datos in situ se almacenan bajo un esquema de *entorno share-nothing* para facilitar la *replicación* y *particionamiento* de sus datos, mientras que los streams utilizan un esquema de *write-ahead logs* para lograr la inmutabilidad y secuencia de ordenación de los datos o eventos que ingresan al sistema. Existe otra capa de extracción y carga de datos, en los cuales los datos se limpian, combinan, comprimen y *replican* en preparación para su análisis. En la capa de análisis, los datos se *procesan por lotes* para los datos in situ o en *tiempo real* para los streams de datos. Existe una capa de trabajos que permite programar la ejecución de procesamiento por lotes de los modelos e incluso su análisis con técnicas de machine learning. Finalmente existe una capa de visualización de los resultados a través de diferentes tipos de interfaces de usuario. Esta arquitectura de referencia utiliza principalmente los casos de uso de Facebook [140], LinkedIn [128], Twitter [130], Netflix [141] y BlockMon [164] para su armado.

Atributo de calidad	Escenario de uso general	Tácticas principales
Escalabilidad	Almacenar grandes volumenes de datos.	*Particionamiento de datos* para distribuir grandes cantidades de datos en varias instancias.
Escalabilidad	Procesar grandes volumenes de datos.	*Procesamiento en lotes* para los datos in situ.
Eficiencia	Alta frecuencia en el análisis y publicación de resultados.	*Procesamiento en tiempo real* para los datos en movilidad.
Disponibilidad	Recuperar ante caídas en el procesamiento distribuido de datos.	*Write ahead logs* para secuenciar y ordenar los datos y eventos que ingresan al sistema bajo *entornos de share-nothing*.
Disponibilidad	Evitar pérdida de información por caídas del sistema.	*Replicación de datos* para tener varias instancias de los datos disponibles.

Tabla 15 Tácticas principales para arquitectura de referencia de Pääkkönen

5.4. Resultados y discusión

Las arquitecturas anteriormente presentadas plantean varios escenarios de uso característicos de big data. Estos escenarios caracterizan atributos de calidad comunes que son resueltos a través de tácticas de arquitectura de software. Varios de los requerimientos identificados y descriptos anteriormente son propios de sistemas distribuidos y son necesarios para poder soportar las características de volumen, velocidad y variedad. En particular, la distribución de sistemas impone ciertas restricciones que son descriptas bajo el teorema de CAP de Brewer [34]. Cuando una partición (P) ocurre con la posibilidad de pérdida de mensajes, el sistema debe elegir entre mantener la consistencia (C) (todos los lectores leen el mismo dato) sobre disponibilidad (A) (todas las consultas deben responderse en un tiempo determinado). El teorema de PACELC [165] provee una interpretación práctica del CAP relacionado con la latencia de los sistemas. Si una partición (P) ocurre, el sistema debe balancear la disponibilidad (A) con respecto a la consistencia (C). En otro caso (E), si no existen particiones, el sistema debe balancear la latencia (L) con respecto a la consistencia (C). En ambos casos de distribución se debe balancear la disponibilidad y velocidad de respuesta del sistema frente a mantener una consistencia estricta y lineal de los datos. Estos son elementos que un arquitecto de software tiene que balancear de acuerdo con las necesidades propias del negocio. Una forma de facilitar el balance es a través del entendimiento más profundo de las tácticas de arquitectura para cada atributo de calidad como se describe a continuación.

5.4.1. Escalabilidad

Los dos escenarios principales en big data para la escalabilidad horizontal incluyen poder distribuir el almacenamiento y procesamiento de grandes volúmenes de datos en una variedad de formatos como se muestra en la Tabla 16.

Escenario de uso	Táctica
Gestionar grandes volúmenes de datos.	*Particionamiento de datos*
Procesar grandes volúmenes de datos.	*Procesamiento por lotes* *Entornos shared-nothing*

Tabla 16 Tácticas de escalabilidad

5.4.1.1. Particionamiento de datos

El objetivo del particionamiento es distribuir los datos y sus consultas en forma equitativa entre los diferentes nodos del sistema. Esto genera el beneficio de poder escalar y distribuir la carga del sistema entre las diferentes particiones. Si no se logra una distribución equitativa pueden existir nodos, conocidos como "hot spots", con una mayor carga operativa perdiendo así los beneficios del particionamiento. De aquí radica la importancia en la elección del esquema de particionamiento para lograr la mejor distribución de los datos. A continuación, se resumen dos esquemas utilizados para el particionamiento y la elección de las claves para identificar los elementos y realizar la distribución de los datos.

1. **Particionamiento por rangos de clave.**

   Para cada partición se generan rangos de claves ordenadas que facilitan la búsqueda de los datos al realizar las consultas. Sin embargo, son propensas a generar "hot spots"[8] si la aplicación accede regularmente a ciertos rangos consecutivos.

2. **Particionamiento por hash.**

   La distribución de los datos se realiza a través de una función de hash. Este método elimina el ordenamiento secuencial de las claves haciendo menos eficiente la búsqueda ordenada, pero asegura una mejor distribución equitativa de las particiones.

[8] Cuando las particiones no son uniformemente distribuidas pueden generarse particiones con mayor demanda denominadas “hot spots”.

La evolución en los datos del sistema puede generar cambios en las particiones debido al incremento del volumen de datos en el tiempo. Esto implica que se deben volver a balancear las particiones para acomodar la nueva distribución de los datos.

5.4.1.2. Entorno share-nothing

Los entornos shared-nothing [166] permiten tener nodos independientes y auto-suficientes, evitando tener un único punto de contención del sistema y excesiva sincronización entre los nodos.

La clasificación de los entornos distribuidos en términos de su arquitectura puede realizarse en diferentes niveles. La clasificación de Stonebraker [166] permite distinguir tres grandes aproximaciones (Figura 21): shared-everything, shared-disk y shared-nothing.

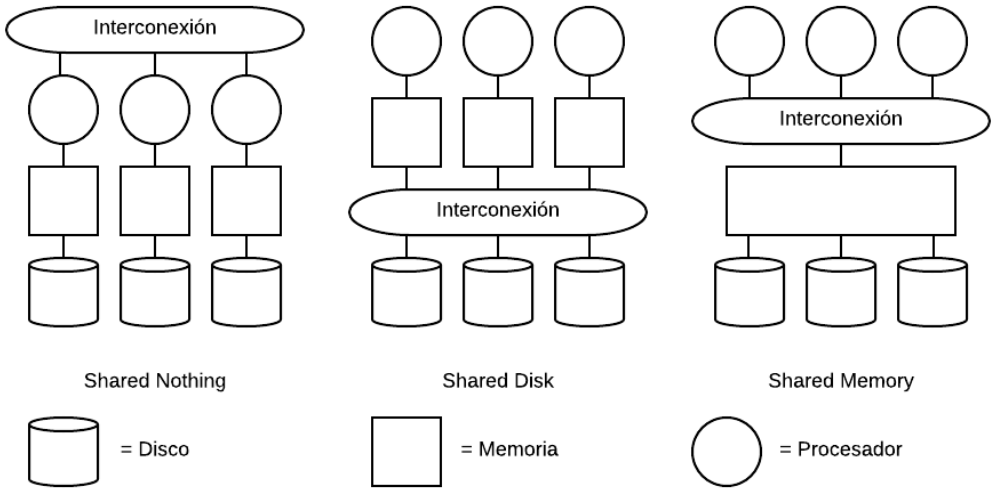

Figura 21 Entornos sistemas distribuidos

Los entornos shared-everything comparten todos los recursos de procesamiento, memoria y almacenamiento. Esto facilita la comunicación entre los componentes, pero a su vez genera un alto acoplamiento que dificulta la escalabilidad del sistema. Los sistemas shared-disk comparten el recurso de almacenamiento el cual se vuelve un único punto de contención. Sin embargo, estos esquemas se vuelven prohibitivos bajo sistemas de big data que necesitan escalar los datos y su procesamiento. Los entornos shared-nothing se vuelven útiles en estos contextos donde se necesita una alta escalabilidad y disponibilidad de los datos. En estos casos es necesario una red de alta velocidad para interconectar los diferentes nodos para conseguir las latencias esperadas.

La construcción de un sistema shared-nothing se puede dividir en los siguientes pasos [167]:

1. **Particionar los nodos:** Esto implica generar un particionamiento del sistema en nodos independientes y autosuficientes. Las particiones pueden ser a nivel de datos o funciones del sistema.
2. **Distribución de los nodos:** Una vez generada las particiones se deben distribuir las instancias de datos y procesamiento dentro de cada nodo.
3. **Configuración del balanceo de carga:** Dada la elección de la partición se debe configurar la distribución y balanceo de la carga del sistema entre sus diferentes nodos.

Este proceso hay que repetirlo en casos que se quiera volver a particionar el sistema de forma diferente.

5.4.1.3. Procesamiento por lotes

El procesamiento por lotes permite la ingesta de un gran volumen de datos a través de un proceso que generalmente se programa para ejecutarse periódicamente. Un sistema de procesamiento por lotes permite procesar lotes de datos a través de una serie de procesos o trabajos sin necesidad de intervención manual. En general los procesos son ejecutados regularmente y pueden tardar un tiempo considerable (lo que los distingue del procesamiento en tiempo real). La principal medida de eficiencia del procesamiento por lotes es el tamaño (o volumen) de datos que se puede procesar bajo cierto tiempo determinado.

Un modelo de programación muy utilizado en estos esquemas es el MapReduce [54]. El proceso se puede visualizar como funciones de transformación de entradas en salidas bajo dos procesos principales de mapeo y reducción.

Mapeo: El mapeo es realizado una vez por cada registro de entrada después de extraer su clave y valor a ser procesado. Por cada entrada se genera un número de pares nuevos de clave y valor a ser ordenados y reducidos. Este proceso no guarda el estado de procesamiento de entradas anteriores por lo cual se puede realizar en paralelo y de forma independiente. El proceso de mapeo puede ser resumido bajo la transformación Map(k1, v1) => List(k2, v2).

Reducción: La nueva lista de clave y valor producidos en el proceso de mapeo son agrupados bajo las mismas claves para ser reducidos. Cada proceso de reducción procesa todos los valores asociados a una clave para generar el resultado final. Este proceso en su conjunto genera una

nueva lista de valores reducidos por cada una de las claves procesadas. El proceso de reducción puede ser resumido bajo la transformación Reduce(k2, list(v2)) -> list(k2, v3).

Estos dos procesos permiten procesar grandes volúmenes de datos en forma distribuida y paralela como se describe en el proceso más detallado de la Figura 22.

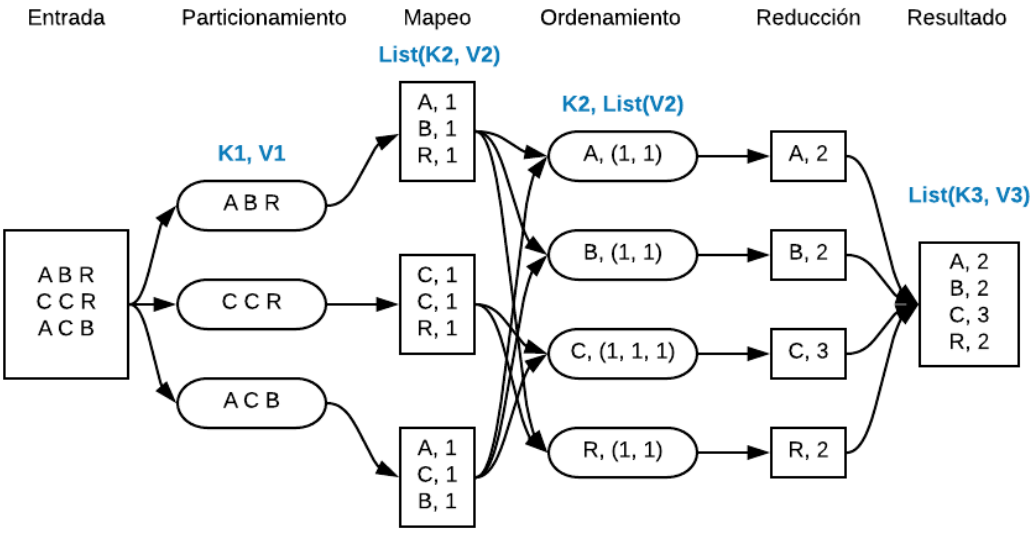

Figura 22 Proceso MapReduce

1. **Particionar:** Se leen un conjunto de entradas que se dividen en registros con claves a ser procesados por cada valor.
2. **Mapear:** Se utiliza el proceso de mapeo para procesar la clave y valor de cada registro.
3. **Ordenar:** Se agrupan ordenadamente los registros por clave para ser procesados por cada reductor.
4. **Reducir:** Se utiliza el proceso de reducción para iterar y procesar cada registro ordenado por su clave para obtener el resultado final.

5.4.2. Eficiencia

Podemos analizar las latencias en soluciones de big data del punto de vista de la gestión de sus datos, así como de los tiempos utilizados en su procesamiento y análisis. La Tabla 17 describe las diferentes tácticas de eficiencia de acuerdo con el tipo de respuesta que queremos lograr.

Escenario de uso	Táctica
Alta frecuencia de generación y lectura de datos	*Consistencia eventual*
Alta frecuencia en el análisis y publicación de resultados.	*Procesamiento en tiempo real*

Tabla 17 Tácticas de eficiencia

5.4.2.1. Consistencia eventual

En sistemas distribuidos, bajo múltiples nodos y particiones, el teorema del CAP nos sugiere que debemos balancear entre disponibilidad y consistencia [168]. Para lograr sistemas altamente disponibles debemos entonces relajar su nivel de consistencia. La introducción del concepto de consistencia eventual [35] permite sincronizar los datos de las particiones de forma eventual, incluso frente a fallas que puedan ocurrir entre las particiones. La sincronización de cambios entre réplicas no es instantánea y dependiendo del nivel de consistencia que se quiera lograr existen diferentes modelos.

Un modelo de consistencia es un contrato entre el almacenamiento de datos y el proceso que lo consume. Normalmente cuando un proceso consulta un dato, la operación espera obtener la última versión escrita del elemento. En caso de existir múltiples réplicas de los datos puede generarse una inconsistencia en las versiones replicadas en el tiempo.

Existen distintos modelos o niveles de consistencia de acuerdo con el balance que se quiera lograr entre la latencia de las operaciones de lectura/escritura y la consistencia de los datos replicados. A continuación, se describen los modelos principales (Figura 23) [169].

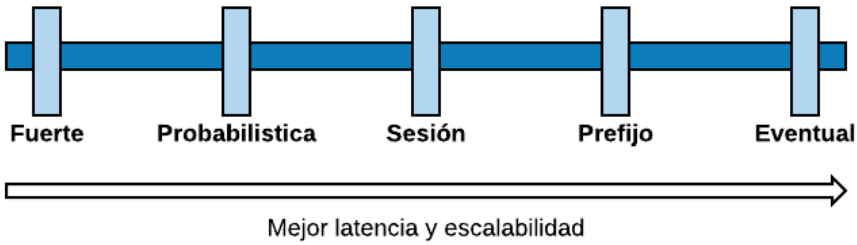

Figura 23 Modelos de consistencia

- **Consistencia fuerte:** Este modelo asegura la linealidad de la información. Las lecturas garantizan que las escrituras sean visibles una vez que son confirmadas en consenso por la mayoría de las réplicas.

- **Consistencia probabilística:** Este nivel permite calcular lo eventual y consistente de los datos con un modelo probabilístico. Se utilizan dos métricas para predecir el nivel de consistencia eventual. Por un lado, el tiempo de consistencia se calcula con la probabilidad de leer un elemento después de un tiempo t de ser escrito. Esto permite contestar el grado de eventualidad del modelo. Por otro lado, la probabilidad de consultar una versión anterior K veces menor a la última versión escrita hasta el momento, permite estimar la consistencia del modelo. La combinación de estas dos métricas permite generar un estimativo del tiempo total necesario para replicar cada versión del elemento escrito.
- **Consistencia por sesión:** Este modelo asegura una consistencia local por sesión de usuario y no global como las dos anteriores. Este nivel asegura lecturas monotónicas, donde cada lectura obtendrá versiones del elemento más recientes a las lecturas anteriores. Las escrituras mantendrán un orden consecutivo dentro de la misma sesión de usuario, lo que se conoce como escrituras monotónicas. Finalmente, las últimas escrituras siempre serán obtenidas en posteriores lecturas dentro de la misma sesión, pero no es garantizado entre diferentes sesiones.
- **Consistencia por prefijo:** La consistencia por prefijo asegura que las lecturas son obtenidas en el mismo orden que ocurren las escrituras. Si las escrituras se generan en el orden A, B, C, entonces un cliente debe seguir viendo ese mismo orden en sus lecturas.
- **Consistencia eventual:** Es la forma más débil de consistencia, pero se obtienen las mejores latencias de escrituras y lecturas. Se garantiza qué en la ausencia de más escrituras, las réplicas convergen eventualmente en el tiempo, pero no se asegura el orden que el cliente obtiene los valores escritos.

5.4.2.2. Procesamiento en tiempo real

Mientras que los procesamientos por lotes necesitan de todos los registros de entrada para generar los resultados, existe el concepto de streams para lograr el procesamiento en tiempo real. Los streams son un flujo de datos que se obtienen incrementalmente en el tiempo y pueden no tener un final determinado. En el contexto del procesamiento de streams, a un registro de entrada se le conoce como un evento. El evento es un objeto inmutable que contiene los detalles de un suceso en el tiempo. El evento es generado por un productor y puede ser procesado por

múltiples consumidores. Generalmente los eventos relacionados se pueden agrupar en tópicos donde los consumidores pueden subscribirse para consumirlos.

Para sincronizar el consumo de los distintos subscriptores se debe utilizar alguna estructura de almacenamiento de los eventos en el tiempo con sus correspondientes notificaciones. Este tipo de requerimientos pueden ser implementados con sistemas de mensajería, donde el consumidor genera un evento en forma de mensaje y este es notificado a los diferentes consumidores. Algunos desafíos en estos sistemas son la sincronización de procesamiento entre el productor y consumidor, y el manejo de la pérdida de mensajes frente a caídas de los productores y/o consumidores. Una forma de resolver estos casos es a través de colas o buffers que permiten agrupar los eventos producidos en diferentes tópicos. Estos tópicos permiten agrupar, persistir y retener los mensajes para recuperar posibles caídas y retrasos de los subscriptores en el consumo de sus mensajes. Incluso la retención indefinida de mensajes suele utilizarse para poder agregar fácilmente futuros subscriptores que quieran procesar la totalidad de los eventos dentro de un tópico en particular.

5.4.3. Modificabilidad

La característica de variedad de big data implica gestionar una diversidad de formatos de datos. Por un lado, existen los datos estructurados que pueden ser descriptos bajo un esquema y estructura fija. Por otro lado, los datos semi-estructurados surgen cuando no existe una estructura rígida (como la Web) y generalmente combinan diferentes orígenes heterogéneos de datos. Los datos semi-estructurados son caracterizados por una estructura flexible y autodescriptiva. Por último, encontramos los datos no estructurados, como generalmente son los archivos de multimedia (texto, video, imágenes), que no contienen ninguna estructura o esquema asociado. La alta diversidad de formatos genera el desafío de persistir y gestionar los datos bajo una misma solución.

Escenario de uso	**Táctica**
Múltiples formatos de datos (estructurados, semi-estructurados y no estructurados)	*Persistencia políglota*

Tabla 18 Tácticas de modificabilidad

5.4.3.1. Persistencia políglota

Las bases de datos relacionales permiten unificar e integrar los datos de varias aplicaciones en una única fuente de datos. En este escenario, la base de datos puede actuar como integradora de múltiples aplicaciones, usualmente desarrollada por diferentes equipos, bajo un almacenamiento común. Esto facilita la consistencia y transaccionalidad de las operaciones bajo las propiedades denominadas ACID [170].

- Atómica: Todas las operaciones de una transacción se realizan en forma atómica. Si sucede una falla en una de las operaciones se cancela la transacción en su conjunto. Esto permite asegurar la integridad de los datos.
- Consistente: Existe una consistencia lineal donde cada escritura queda disponible inmediatamente para posteriores lecturas y consultas.
- Aislada: El aislamiento transaccional permite asegurar que las operaciones concurrentes sean sincronizadas correctamente.
- Durable: Cuando se completa una transacción esta debe persistir incluso ante posibles fallas y caídas del sistema.

Sin embargo, la variedad de formatos implica complejizar el diseño del modelo común para acomodar las diferentes estructuras y esquemas heterogéneos de datos. Si una de las aplicaciones quiere realizar modificaciones al modelo, esto tiene un impacto directo sobre las otras aplicaciones que la utilizan. Además existe la dificultad de evolucionar y migrar los esquemas durante el ciclo de vida del producto [171]. La desventaja de los modelos con esquemas rígidos es que no tienen la flexibilidad de aplicar migraciones diferidas [172] para evitar interrupciones [173] en el servicio de la aplicación. Por otro lado, la necesidad de manejar un gran volumen de datos es limitado en contextos ACID donde solo se puede escalar verticalmente (sobre un único nodo). Esto se debe a que se prioriza la fuerte consistencia sobre la disponibilidad bajo múltiples particiones que permitan escalar horizontalmente.

Para resolver estos desafíos surgen las bases de datos NoSQL con las propiedades denominadas BASE [174] (Basic availability, Soft-state, Eventual consistency). Estas propiedades priorizan la disponibilidad y escalabilidad a través de la redundancia y particionamiento en múltiples réplicas. Esto implica que las réplicas no siempre serán consistentes, pero podrán escalar horizontalmente y sincronizarán eventualmente. Existe una variedad de base de datos NoSQL que se pueden categorizar de acuerdo a su modelo de datos [175] como resume la Tabla 19.

Modelo de datos	Descripción	Tipo de estructura
Clave-valor	Consiste en un conjunto de pares de clave-valor únicos. Su simple estructura permite operaciones de creación, lectura, borrado y actualización. No impone ningún esquema de datos para sus valores, lo que restringe las consultas unicamente por clave.	No estructurado
Documental	Consiste en un almacenamiento clave-valor pero restringe los valores a algún formato semi-estructurado como documentos JSON. Esta restricción de estructura agrega una mayor capacidad de consulta sobre el propio documento.	Semi-estructurado
Familia de columnas	Consiste en familias de columnas ordenadas por fila. Cada columna tiene su propia clave de acceso así como cada una de sus filas. La estructura está optimizada para almacenar grandes cantidades de columnas por fila.	Estructurado
Grafos	Permite almacenar entidades con sus relaciones. Las entidades son nodos o vértices de un grafo con propiedades. Las relaciones se representan como aristas con propiedades que conectan los diferentes vértices. La estructura del grafo está optimizada para poder navegar las diferentes relaciones y sus nodos asociados.	Semi-estructurado

Tabla 19 Tipos de base de datos NoSQL

La *persistencia políglota* [176] propone utilizar múltiples modelos de datos, bajo una misma solución, de acuerdo a cada necesidad particular de persistencia. Esto deriva del concepto de utilizar la herramienta adecuada al problema a resolver. Esta diversidad de modelos de datos genera un desafío adicional al practicante que debe poder elegir la mejor opción de acuerdo con su contexto de uso. Existe un interés en la comunidad [170] [175] para evaluar la elección del modelo de datos adecuado según el contexto de uso. Un ejemplo de esto es la base de conocimiento del SEI [82] para la evaluación de base de datos distribuidas denominado QuABaseBD[9] (Quality Attributes at Scale Knowledge Base). Por otro lado, existen base de datos multi-modelos [177] que agrupan los beneficios de diferentes modelos de datos bajo una misma base de datos.

5.4.4. Disponibilidad

En los sistemas distribuidos la disponibilidad típicamente se logra a través de la redundancia de sus componentes. La replicación permite mantener varias copias de los componentes y datos en diferentes nodos (potencialmente distribuidos geográficamente). Ante la falla o caída de

[9] https://quabase.sei.cmu.edu

algún nodo, la replicación permite la redundancia para lograr servir los datos o componentes desde los nodos operativos restantes.

Escenario de uso	Táctica
Recuperar ante fallas en el almacenamiento, procesamiento y comunicación de la plataforma	*Write ahead logs* *Semánticas de llamadas remotas*
Tolerar fallas en el almacenamiento, procesamiento y comunicación de la plataforma	*Replicación de datos* *Replicación funcional de procesamiento*

Tabla 20 Tácticas de disponibilidad

5.4.4.1. Write-Ahead Log

Una variedad de bases de datos ha utilizado el mecanismo de write-ahead logs como fuente para recuperarse frente a fallas durante la caída de sus transacciones. Un log [178] es una secuencia inmutable de registros ordenadas temporalmente donde solo se pueden agregar nuevos datos al final de la secuencia. Los nuevos registros se añaden al final de la secuencia y el contenido es leído desde el comienzo en forma secuencial hasta el final del log. Cada entrada del log contiene un número único secuencial para lograr un ordenamiento de los registros como se muestra en la Figura 24.

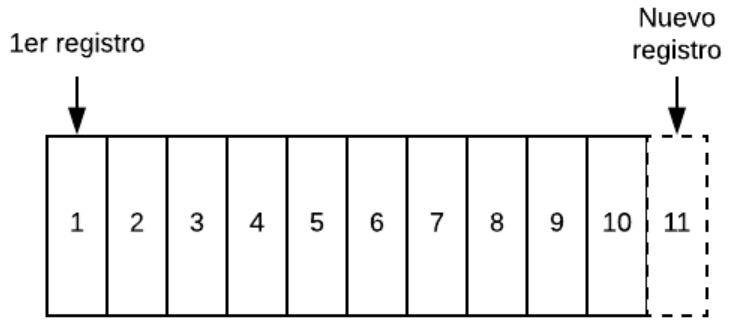

Figura 24 Estructura de un log

Esta ordenación asegura procesamientos determinísticos incluso bajo esquemas de alta distribución y replicación. Al partir de un mismo estado y procesar los registros bajo la misma secuencia de eventos se asegura un mismo resultado cada vez. Esto facilita la distribución del procesamiento de un gran volumen de datos, permitiendo utilizar un esquema de productor/suscriptor para analizar el log. Los productores generan el log en forma secuencial a partir de los eventos o datos generados en el sistema. Esto facilita que cada subscriptor pueda

procesar los registros a su propio ritmo. Cada subscriptor utiliza el índice dentro de la secuencia del log para identificar su etapa de procesamiento. Ante posibles fallas y bajo la propiedad de inmutabilidad del log, los subscriptores pueden reiniciar su procesamiento desde el lugar donde se produjo la falla.

Este esquema permite guardar todos los eventos del sistema y reproducir el estado del sistema en cada momento. Si el procesamiento se realiza en tiempo real, el log se transforma en un flujo (stream) de datos que se actualiza en tiempo real con cada nuevo registro producido. Este mecanismo utiliza una única estructura de datos simple, escalable y sincronizable entre múltiples productores y subscriptores. Agregar otra forma de análisis de los eventos implica agregar otro subscriptor con la nueva lógica de procesamiento.

5.4.4.2. Semánticas de llamadas remotas

En sistemas distribuidos también hay que lidiar con fallas que puedan producirse durante las llamadas remotas. Existen diferentes semánticas de llamadas [179] entre componentes del sistema para enviarse mensajes. Idealmente queremos garantizar el envío único del mensaje a través de cada llamada. Sin embargo, esto no es sencillo de conseguir en sistemas distribuidos donde pueden existir diferentes fallas durante la comunicación y envío de mensajes. Es por esto que existen diferentes semánticas de llamadas como se describen a continuación.

- **At most once:** En este modelo el envío del mensaje se realiza una única vez. Esto implica que si existen fallos puede que no se envíe el mensaje correctamente. Este modelo se utiliza cuando la pérdida ocasional de mensajes no es un problema.
- **At least once:** Este modelo asegura que el receptor recibe al menos una vez el mensaje. Para garantizar la comunicación, el emisor reintenta el envío del mensaje hasta que el receptor confirma su correcta recepción. Esta aproximación puede ser utilizada cuando la operación a realizar es idempotente, ya que no podemos asegurar la unicidad de la llamada por reintentos.
- **Exactly once:** Este es el caso ideal donde se consigue enviar el mensaje exactamente una única vez. Sin embargo, esto no es posible de lograr en sistemas distribuidos donde pueden existir diferentes tipos de fallas durante la comunicación. El resultado final puede ser simulado a través de aspectos transaccionales e idempotentes de las operaciones bajo esquemas de at least once.

5.4.4.3. Replicación de datos y funcional

La replicación es una forma de lograr la redundancia de los diferentes componentes de una arquitectura para enmascarar sus posibles fallas de sistema. La replicación implica tener varias copias de los mismos datos o funciones distribuidos en múltiples nodos denominadas réplicas. Existen diferentes beneficios de la replicación como pueden ser:

- Reducir la latencia manteniendo las copias cercanas a los consumidores finales bajo esquemas de distribución geográficamente.
- Incrementar la disponibilidad permitiendo al sistema seguir funcionando incluso ante fallas que puedan suceder en un número determinado de réplicas.
- Aumentar el rendimiento del sistema al poder escalar la cantidad de nodos que sirven datos o realicen procesamiento.

La dificultad de la replicación radica en los cambios que se deben propagar para mantener las réplicas sincronizadas. Existen diferentes algoritmos de replicación en la literatura para lograr este objetivo. A continuación se describen los tres principales [180].

1. **Replicación basada en un líder (Leader-based replication)**
   En este caso una de las réplicas es designada como el líder (también conocido como master o primario). Todas las escrituras son enviadas al líder para ser persistidas. El resto de las réplicas son seguidores y reciben los cambios a sincronizar a través del líder. Cuando un cliente quiere leer los datos lo puede hacer desde el líder, o alguno de los seguidores, en cambio las escrituras son siempre realizadas a través del líder.
2. **Replicación multi líder (Multi-leader replication)**
   Este modelo es una extensión del anterior, donde existen múltiples líderes que pueden recibir escrituras. Esto implica que los líderes deben poder sincronizarse con los otros líderes del sistema. Existen diferentes topologías para realizar estas sincronizaciones como puede ser el cirular, estrella o entre todos los líderes. Esto es de utilidad en contextos de centros de datos distribuidos geográficamente, donde cada centro tiene su propio líder además de sus seguidores. Esta aproximación agrega la complejidad de resolver los conflictos de escritura entre líderes. Existen diferentes formas de resolución de conflictos, aunque ninguna óptima ya que depende del dominio de aplicación. Esto hace que muchas herramientas permitan implementar código particular para resolver dichas situaciones particulares.

3. **Replicación sin líder (Leaderless replication)**

   En este modelo no existe el concepto de líderes y las escrituras y lecturas se realizan a un número determinado de nodos en forma simultánea. Para asegurarse la consistencia de datos el número de escrituras y lecturas deben solaparse para asegurar que las lecturas se realizan sobre al menos un nodo con las últimas versiones de los datos.

5.1. Amenazas a la validez

Los resultados de la evaluación de las arquitecturas de software pueden ser afectados por una serie de amenazas a la validez [121] que son analizadas a continuación.

5.1.1. Validez interna

La validez interna refiere a la conexión entre lo observado y las explicaciones propuestas [122]. Esto asegura que las conclusiones obtenidas sean verdaderas. La evaluación de las arquitecturas se basa en el correcto entendimiento de los requerimientos y escenarios de uso del problema que las arquitecturas buscan resolver. Por lo tanto, se buscaron organizaciones que publiquen en artículos o blogs sus desafíos y soluciones propuestas con la comunidad. Esto restringe el estudio a la información disponible y publicada por estas empresas. Sin embargo, notamos que muchas de estas empresas están dispuestas a compartir su conocimiento, siendo la cantidad de tecnologías desarrolladas internamente y posteriormente compartidas como open source una muestra de ello.

5.1.2. Validez externa

La validez externa se refiere a la posibilidad de generalización de los resultados [122]. Dada la cobertura de los diferentes tipos de arquitecturas (industriales, teóricas y de referencia) permite tener una variedad de casos analizados. Sin embargo, puede que el número analizado por cada tipo de arquitectura no sea lo suficiente para lograr generalizar completamente los hallazgos. Es parte de la investigación futura extender la evaluación a un mayor número de arquitecturas que permitan ampliar el catálogo de escenarios y tácticas arquitectónicas.

5.1.3. Validez de conclusión

La validez de conclusión se refiere a la relación entre el tratamiento y el resultado final observado [122]. Una amenaza a la validez de conclusión se puede dar en la elección de las

arquitecturas analizadas. Para disminuir esta amenaza se validaron con otro investigador el conocimiento y representatividad de las arquitecturas de software dentro de la comunidad de arquitectos. Otra posible amenaza es la selección de tácticas de arquitectura. Para esto se utilizó un modelo y método que permitiera derivar tácticas a partir de escenarios de uso concretos identificados en las arquitecturas. Una mejora en este proceso es la utilización de marcos de trabajo de razonamiento (reasoning frameworks) [70] para detallar la relación entre los atributos de calidad y las tácticas de arquitectura, quedando esto para una futura investigación.

5.1.4. Validez del constructo

La validez del constructo asegura que la construcción del estudio esté relacionada con el problema de investigación [122]. Una posible amenaza a la validez del constructo puede surgir en la adaptación del método de evaluación utilizado. ATAM fue originalmente pensado para evaluar diferentes opciones de arquitectura bajo un dominio particular de negocio. En este caso se adaptó para analizar un dominio más general, como lo es big data, bajo una variedad de tipos de arquitectura de software. Creemos que la generación de los escenarios de uso generales de big data (volumen, velocidad y variedad), y los escenarios de uso concretos extraídos de las organizaciones, logran definir un dominio de aplicación para una correcta aplicación de ATAM.

6. Conclusiones

Las soluciones de software pueden clasificarse como big data cuando la capacidad de procesamiento de las bases de datos tradicionales no es suficiente y se vuelve imperativo el uso de tecnologías alternativas. Es por esto por lo que al diseñar arquitecturas de software bajo contextos de big data, las aproximaciones tradicionales no son suficientes para resolver dichos desafíos.

Una forma de visualizar está complejidad es a través de los atributos de calidad del problema a resolver. La definición de las características de calidad como requerimientos no funcionales son centrales para las decisiones de diseño de una arquitectura de software. Para lograr caracterizar los atributos de calidad es posible especificar escenarios que ejemplifiquen los requerimientos no funcionales a satisfacer.

El presente trabajo busca conectar los escenarios de uso y atributos de calidad encontrados bajo los contextos de big data con las tácticas de arquitectura que guíen su resolución. Del análisis de las descripciones de las arquitecturas identificadas en el mapeo sistemático se pueden elaborar escenarios de uso con problemáticas comunes. Muchos de los desafíos son similares a los encontrados en sistemas distribuidos [181] pero a mayor escala determinados por las 3Vs (volumen, variedad y velocidad) [3]. Esto tiene un impacto directo en las decisiones de diseño que un arquitecto de software debe contemplar frente a situaciones similares. Los sistemas de big data son inherentemente distribuidos, por lo que sus arquitecturas deben lidiar explícitamente con las fallas, latencias de comunicación, concurrencia, consistencia y replicación de sus múltiples nodos y componentes distribuidos. Al crecer y replicar el sistema a miles de nodos distribuidos geográficamente estos desafíos aumentan con la mayor probabilidad de fallas en el hardware y software [182].

Para superar dichos desafíos, es necesario crear arquitecturas resilientes que contemplen aspectos de replicación y particionamiento en las diferentes capas de la solución. Se vuelve necesario distribuir el estado de la aplicación para asegurar la disponibilidad del sistema frente a fallas en el almacenamiento o comunicación entre sus nodos. Las réplicas deben mantenerse estricta o eventualmente consistentes y lidiar con inconsistencias bajo situaciones de concurrencia. Por otro lado, los componentes que procesan el estado de la aplicación también deben ser replicados para lograr procesar el gran volumen de datos distribuidos y ser tolerantes ante posibles fallas del entorno.

Los atributos propios de los contextos de big data generan la necesidad de tener arquitecturas complejas de varias capas. Esto se puede observar en las arquitecturas de referencia que contienen una variedad de capas (cada una con sus propios desafíos tecnológicos) para lograr la persistencia, extracción, procesamiento, análisis, transformación y visualización de datos con características de big data. Por lo tanto, es crucial el entendimiento y diseño de cada componente para conseguir una solución integral de software. Esto implica muchas veces un gran desafío para el practicante que debe manejar los diferentes conceptos y tecnologías propias de cada capa.

Una buena forma de capturar estas decisiones de diseño es a través de las tácticas y patrones de arquitectura. Estos elementos son centrales para poder hacer un análisis acertado en las decisiones de diseño y lograr que tengan un impacto positivo sobre los requerimientos propios de big data. Es por esto que se entiende que las tácticas de arquitectura son esenciales para guiar al arquitecto en la toma de decisiones de diseño en la solución final. Esto se vuelve especialmente importante en contextos de big data donde el estudio identifica la complejidad de los requerimientos y la necesidad de utilizar tácticas particulares para su correcto diseño e implementación.

El punto de partida de esta línea de investigación fue la identificación y categorización de las arquitecturas de software de big data en un estudio de mapeo sistemático. El segundo estudio profundiza en la evaluación de dichas arquitecturas para identificar, describir y discutir el impacto de un conjunto de tácticas arquitectónicas comunes de big data. Ambos estudios intentan acercar el conocimiento de las arquitecturas de software y tácticas al practicante de forma de facilitar el proceso de diseño arquitectónico en soluciones de software de big data.

6.1. Resultados obtenidos

A continuación se realiza un resumen de los resultados obtenidos en la tesis para cada una de las preguntas de investigación:

- Q1 busca identificar las propuestas de arquitectura de software existente en la literatura. Como resultado se identificaron un total de 90 propuestas de arquitecturas de software bajo contextos de big data. Las propuestas se pueden clasificar, para responder Q2, en 38% industriales, 53% teóricas y 9% de referencia. Es posible decir que las mayorías de arquitecturas de software especifican múltiples capas (61%), aunque existe un gran

número que se centra en el almacenamiento de datos exclusivamente (21%), en el análisis de datos (10%) y en el procesamiento de datos (8%).

- Q3 y Q4 buscan explorar las características de calidad comunes en big data. Durante el estudio se descubren similitudes a los escenarios de uso de los sistemas distribuidos, pero a mayor escala de velocidad, variedad y volumen de datos. Siendo los atributos de calidad de disponibilidad, eficiencia, escalabilidad y modificabilidad los cuatro de mayor impacto en las arquitecturas de software analizadas durante la tesis.
- Q5 y Q6 buscan describir los tipos de decisiones necesarias para satisfacer los atributos de calidad propios de big data. La tesis identifica y describe 10 tácticas de arquitectura (Particionamiento de datos, Procesamiento por lotes, Entornos shared-nothing, Procesamiento en tiempo real, Semánticas de llamadas remotas, Write-ahead logs, Replicación de datos, Replicación funcional, Consistencia eventual, Persistencia políglota) de software para resolver 7 escenarios de uso generales de big data.

6.2. Publicaciones asociadas a este trabajo

Durante el transcurso de la investigación presentada en la tesis se elaboraron dos artículos. El primero se titula “Estudio de Mapeo Sistemático sobre Arquitecturas de Software para Big Data” y fue aceptado en la XX Congreso Iberoamericano en Ingeniería de Software (CIbSE 2017) y presentado en Buenos Aires (Argentina) el día 23 de Mayo de 2017 [183]. Esta publicación resume la primera parte de la investigación llevada a cabo para esta tesis.

El segundo artículo lleva el título “Análisis de Tácticas en Arquitecturas de Software para Big Data”. Este trabajo resume todo lo relacionado con el análisis de las tácticas de arquitecturas de big data. El artículo será enviado a una conferencia arbitrada a definir relacionada con el área de arquitecturas de software.

6.3. Líneas futuras de investigación

En esta sección se presentan las posibles líneas de investigación a seguir, tomando como base los aportes realizados en esta tesis.

- Como forma de darle continuidad a la investigación realizada, sobre el modelo inicial de escenarios y atributos de calidad en contextos de big data, se podría profundizar en mayor detalle. Creemos que se puede realizar una revisión sistemática para ampliar el

conocimiento de los atributos de calidad propios de big data. Existe una discusión en la comunidad sobre la diversidad de características de big data [184], lo cual dificulta tener una clara definición de sus atributos de calidad. Por otro lado, existen marcos de trabajo de razonamiento [70] que modelan los comportamientos específicos del sistema para satisfacer los atributos de calidad. Un objetivo sería identificar que modelos de razonamiento pueden ser utilizados para mejorar la caracterización y número de los atributos de calidad de big data.

- La evaluación de un mayor número de arquitecturas de software permitiría extender y generalizar el catálogo de escenarios y tácticas de arquitectura encontradas. Por otro lado, sería interesante utilizar el catálogo en proyectos reales de big data para entender el beneficio y utilidad de las tácticas seleccionadas. Existe un proceso de diseño de big data (BDD) [84] propuesto por el SEI que permite incorporar elementos de diseño, como las arquitecturas de referencia con un catálogo de tácticas y patrones arquitectónicos, para el desarrollo de soluciones de big data. El objetivo sería incorporar los elementos estudiados durante la tesis en el proceso de BDD y evaluar su beneficio para el practicante. Para esto también se podría utilizar el método de ATAM durante el ciclo completo de diseño arquitectónico de un proyecto de big data.
- Las arquitecturas industriales de big data estudiadas evolucionaron desde arquitecturas monolíticas hacia microservicios. Este estilo de arquitectura de software suele implementar algunas de las tácticas analizadas, como la persistencia políglota (ya que permiten tener una base de datos diferente por aplicación o servicio) o el particionamiento funcional (facilitando la replicación y escalamiento de servicios individuales). El objetivo sería comprender el impacto y relación de este estilo de arquitecturas de software sobre los contextos de big data. Esto permitiría extender la línea de investigación actual de tácticas hacia el estudio de patrones o estilos de arquitectura relacionados con big data.

7. Referencias bibliográficas

[1] A. Siddiqa *et al.*, "A Survey of Big Data Management: Taxonomy and State-of-the-Art," *J. Netw. Comput. Appl.*, vol. 71, pp. 151–166, Apr. 2016.

[2] "IBM - What is big data?," 09-Feb-2016. [Online]. Available: http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html. [Accessed: 14-May-2016].

[3] D. Laney, "3D Data Management: Controlling Data Volume, Velocity and Variety," *META Gr. Res. note*, vol. 6, no. 70, p. 1, 2001.

[4] N. Ashish and V. Dan, "Worldwide Big Data Technology and Services Forecast, 2016–2020," *Int. Data Corp.*, 2016.

[5] Signals And System Telecom, "The Big Data Market: 2015 – 2030 - Opportunities, Challenges, Strategies, Industry Verticals & Forecasts," 2015. [Online]. Available: http://www.snstelecom.com/bigdata. [Accessed: 14-May-2016].

[6] "IEEE 1st International Congress on Big Data." [Online]. Available: http://www.ieeebigdata.org/2012/. [Accessed: 14-May-2016].

[7] L. Baresi, T. Menzies, A. Metzger, and T. Zimmermann, "1st International Workshop on Big Data Software Engineering (BIGDSE 2015)," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, 2015, pp. 965–966.

[8] N. H. Madhavji, A. Miranskyy, and K. Kontogiannis, "Big Picture of Big Data Software Engineering: With Example Research Challenges," in *2015 IEEE/ACM 1st International Workshop on Big Data Software Engineering*, 2015, pp. 11–14.

[9] C. E. Otero and A. Peter, "Research directions for engineering big data analytics software," *IEEE Intell. Syst.*, vol. 30, no. 1, pp. 13–19, 2015.

[10] N. Maiden, "Monitoring Our Requirements," *IEEE Softw.*, vol. 30, no. 1, pp. 16–17, Jan. 2013.

[11] W. N. Robinson, "A Roadmap for Comprehensive Requirements Modeling," *Computer (Long. Beach. Calif).*, vol. 43, no. 5, pp. 64–72, May 2010.

[12] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Comput. Surv.*, vol. 46, no. 4, pp. 1–37, Mar. 2014.

[13] A. Tsymbal, "The problem of concept drift: definitions and related work," *Comput. Sci. Dep. Trinity Coll. Dublin*, vol. 106, no. 2, 2004.

[14] F. Villanustre, "Industrial Big Data Analytics: Lessons from the Trenches," in *2015 IEEE/ACM 1st International Workshop on Big Data Software Engineering*, 2015, pp. 1–3.

[15] K. M. Anderson, "Embrace the Challenges: Software Engineering in a Big Data World," in *2015 IEEE/ACM 1st International Workshop on Big Data Software Engineering*, 2015, pp. 19–25.

[16] I. Gorton and J. Klein, "Distribution, data, deployment: Software architecture convergence in big data systems," *IEEE Softw.*, vol. 32, no. 3, pp. 78–85, May 2015.

[17] R. Lu, H. Zhu, X. Liu, J. Liu, and J. Shao, "Toward efficient and privacy-preserving computing in big data era," *IEEE Netw.*, vol. 28, no. 4, pp. 46–50, Jul. 2014.

[18] A. Jacobs, "The pathologies of big data," *Commun. ACM*, vol. 52, no. 8, p. 36, Aug.

2009.

[19] E. Erturk and K. Jyoti, “Perspectives on a Big Data Application: What Database Engineers and IT Students Need to Know,” *Eng. Technol. Appl. Sci. Res.*, vol. 5, no. 5, pp. 850–853, 2015.

[20] F. Shull, “Getting an Intuition for Big Data,” *IEEE Softw.*, vol. 30, no. 4, pp. 3–6, Jul. 2013.

[21] A. Kumar, F. Niu, and C. Ré, “Hazy: Making It Easier to Build and Maintain Big-Data Analytics,” *Commun. ACM*, vol. 56, no. 3, pp. 40–49, Mar. 2013.

[22] X. Xie, J. W. K. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen, “Testing and validating machine learning classifiers by metamorphic testing,” *J. Syst. Softw.*, vol. 84, no. 4, pp. 544–558, Apr. 2011.

[23] R. Yagel, H. (Hans) Hagen, IEEE Computer Society. Technical Committee--Computer Graphics., and SIGGRAPH., *Visualization ’97 : proceedings, October 19-24, 1997, Phoenix, Arizona*. ACM Press, 1997.

[24] J. Manyika *et al.*, “Big data: The next frontier for innovation, competition, and productivity,” *McKinsey Global Institute*, 2011. [Online]. Available: http://www.mckinsey.com/business-functions/business-technology/our-insights/big-data-the-next-frontier-for-innovation. [Accessed: 21-May-2016].

[25] A. De Mauro, M. Greco, and M. Grimaldi, “A formal definition of Big Data based on its essential features,” *Libr. Rev.*, vol. 65, no. 3, pp. 122–135, Apr. 2016.

[26] A. De Mauro, M. Greco, and M. Grimaldi, “What is big data? A consensual definition and a review of key research topics,” in *AIP conference proceedings*, 2015, pp. 97–104.

[27] “12 Big Data Definitions: What’s Yours? - Forbes.” [Online]. Available: http://www.forbes.com/sites/gilpress/2014/09/03/12-big-data-definitions-whats-yours/#7c280fff21a9. [Accessed: 14-May-2016].

[28] J. Rowley, “The wisdom hierarchy: representations of the DIKW hierarchy,” *J. Inf. Sci.*, vol. 33, no. 2, pp. 163–180, Feb. 2007.

[29] Y. Demchenko, Z. Zhao, P. Grosso, A. Wibisono, and C. de Laat, “Addressing Big Data challenges for Scientific Data Infrastructure,” in *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, 2012, pp. 614–617.

[30] O’Reilly Radar Team, *Big Data Now*. O’Reilly Media, 2015.

[31] J. Merino, I. Caballero, B. Rivas, M. Serrano, and M. Piattini, “A Data Quality in Use model for Big Data,” *Futur. Gener. Comput. Syst.*, vol. 63, pp. 123–130, Dec. 2016.

[32] J. Liu, J. Li, W. Li, and J. Wu, “Rethinking big data: A review on the data quality and usage issues,” *ISPRS J. Photogramm. Remote Sens.*, vol. 115, pp. 134–142, Dec. 2015.

[33] A. Fox and E. A. Brewer, “Harvest, yield, and scalable tolerant systems,” in *Proceedings of the Seventh Workshop on Hot Topics in Operating Systems*, 1999, pp. 174–178.

[34] E. Brewer, “CAP twelve years later: How the ‘rules’ have changed,” *Computer (Long. Beach. Calif).*, vol. 45, no. 2, pp. 23–29, Feb. 2012.

[35] P. Bailis and A. Ghodsi, “Eventual consistency today,” *Commun. ACM*, vol. 56, no. 5, p. 55, May 2013.

[36] S. Ceri, M. Negri, and G. Pelagatti, “Horizontal data partitioning in database design,” in

Proceedings of the 1982 ACM SIGMOD international conference on Management of data - SIGMOD '82, 1982, pp. 128–136.

[37] C. C. Aggarwal, *Data Streams*, vol. 31. Boston, MA: Springer US, 2007.

[38] A. Labrinidis and H. V. Jagadish, "Challenges and opportunities with big data," *Proc. VLDB Endow.*, vol. 5, no. 12, pp. 2032–2033, Aug. 2012.

[39] E. Begoli, "A short survey on the state of the art in architectures and platforms for large scale data analysis and knowledge discovery from data," in *Proceedings of the WICSA/ECSA 2012 Companion Volume on - WICSA/ECSA '12*, 2012, pp. 177–183.

[40] M. West, *Developing high quality data models*. Morgan Kaufmann, 2011.

[41] A. Gandomi and M. Haider, "Beyond the hype: Big data concepts, methods, and analytics," *Int. J. Inf. Manage.*, vol. 35, no. 2, pp. 137–144, Apr. 2015.

[42] P. Buneman, "Semistructured data," in *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems - PODS '97*, 1997, pp. 117–121.

[43] A. Madani, O. Boussaid, and D. E. Zegour, "Semi-structured Documents Mining: A Review and Comparison," *Procedia Comput. Sci.*, vol. 22, pp. 330–339, 2013.

[44] E. Dumbill, "We live in an increasingly measured society making sense of big data," *Big Data*, vol. 1, no. 1, 2012.

[45] D. Fisher, R. DeLine, M. Czerwinski, and S. Drucker, "Interactions with big data analytics," *Interactions*, vol. 19, no. 3, pp. 50–59, May 2012.

[46] G. E. Moore, "Cramming More Components Onto Integrated Circuits," *Proc. IEEE*, vol. 86, no. 1, pp. 82–85, Jan. 1998.

[47] T. Honjo and K. Oikawa, "Hardware acceleration of Hadoop MapReduce," in *2013 IEEE International Conference on Big Data*, 2013, pp. 118–124.

[48] S. Ghemawat, H. Gobioff, S.-T. Leung, S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *Proceedings of the nineteenth ACM symposium on Operating systems principles - SOSP '03*, 2003, vol. 37, no. 5, p. 29.

[49] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 2010, pp. 1–10.

[50] G. DeCandia *et al.*, "Dynamo: amazon's highly available key-value store," *ACM SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, p. 205, Oct. 2007.

[51] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, p. 35, Apr. 2010.

[52] A. Corbellini, C. Mateos, A. Zunino, D. Godoy, and S. Schiaffino, "Persisting big-data: The NoSQL landscape," *Inf. Syst.*, vol. 63, pp. 1–23, 2017.

[53] G. Harrison, "Real-Time Big Data With the Lambda Architecture," *Database Trends Appl.*, vol. 28, no. 5, p. 31, 2014.

[54] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[55] H. Chen, R. H. L. Chiang, and V. C. Storey, "Business intelligence and analytics: from

big data to big impact," *MIS Q.*, vol. 36, no. 4, pp. 1165–1188, 2012.

[56] J. Brownlee, "A Tour of Machine Learning Algorithms." [Online]. Available: http://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/. [Accessed: 04-Jun-2016].

[57] A. Mcafee, E. Brynjolfsson, and H. Org, "Big Data: The Management Revolution," *Harv. Bus. Rev.*, vol. 90, no. 10, pp. 60–68, 2012.

[58] J. Ginsberg, M. H. Mohebbi, R. S. Patel, L. Brammer, M. S. Smolinski, and L. Brilliant, "Detecting influenza epidemics using search engine query data.," *Nature*, vol. 457, no. 7232, pp. 1012–1014, Feb. 2009.

[59] N. Askitas and K. F. Zimmermann, "Google Econometrics and Unemployment Forecasting," *IZA Discuss. Pap.*, 2009.

[60] G. Guzmán, "Internet search behavior as an economic forecasting tool: The case of inflation expectations," *J. Econ. Soc. Meas.*, vol. 36, no. 3, pp. 119–167, 2011.

[61] D. Boyd and K. Crawford, "Critical Questions for Big Data: Provocations for a Cultural, Technological, and Scholarly Phenomenon," *Information, Commun. Soc.*, vol. 15, no. 5, pp. 662–679, 2012.

[62] L. Manovich, "Trending: The Promises and the Challenges of Big Social Data," *Debates Digit. Humanit.*, vol. 2, pp. 460–475, 2011.

[63] P. Bourque and R. E. Fairley, *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0*. 2014.

[64] P. B. Kruchten, "The 4+1 View Model of architecture," *IEEE Softw.*, vol. 12, no. 6, pp. 42–50, 1995.

[65] T. R. Gruber, T. R. Gruber, T. R. Gruber, D. M. Russell, and D. M. Russell, "Design Knowledge and Design Rationale: A Framework for Representation, Capture, and Use," *Lab. STANFORD Univ.*, 1991.

[66] Organisation internationale de normalisation, "Systems and software engineering: architecture description.," ISO/IEC/IEEE 42010, 2011.

[67] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice. 3er ed*. Addison-Wesley, 2013.

[68] S. Angelov, J. Trienekens, and R. Kusters, "Software Reference Architectures - Exploring Their Usage and Design in Practice," *Eur. Conf. Softw. Archit.*, pp. 17–24, 2013.

[69] R. Cloutier, G. Muller, D. Verma, R. Nilchiani, E. Hole, and M. Bone, "The Concept of Reference Architectures," *Syst. Eng.*, vol. 13, no. 1, pp. 14–27, 2009.

[70] L. Bass, J. Ivers, M. Klein, and P. Merson, "Reasoning Frameworks," *Carnegie-Mellow Univ. Softw. Eng.*, 2005.

[71] F. Bachmann, L. Bass, and M. Klein, "Deriving Architectural Tactics: A Step Toward Methodical Architectural Design," *Carnegie-Mellow Univ. Softw. Eng.*, 2003.

[72] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and A. Shlomo, *A pattern language : towns, buildings, construction*. Oxford university press, 1977.

[73] P. Avgeriou and U. Zdun, "Architectural patterns revisited – a pattern language," *10th*

Eur. Conf. Pattern Lang. Programs (EuroPlop 2005), Irsee, 2005.

[74] A. Sharma, M. Kumar, and S. Agarwal, "A Complete Survey on Software Architectural Styles and Patterns," *Procedia Comput. Sci.*, vol. 70, pp. 16–28, 2015.

[75] F. Buschmann, *Pattern-oriented software architecture : a system of patterns*. Wiley, 1996.

[76] NIST Group Big Data Public Working, "NIST Big Data Working Group (NBD-WG)," *NIST Special Publication*, 2015. [Online]. Available: https://bigdatawg.nist.gov/home.php. [Accessed: 09-Dec-2017].

[77] NIST Big Data Public Working Group, "NIST Big Data Interoperability Framework: Volume 2, Big Data Taxonomies," *NIST Spec. Publ.*, 2015.

[78] NIST Group Big Data Public Working, "NIST Big Data Interoperability Framework: Volume 5, Architectures White Paper Survey," *NIST Spec. Publ.*, 2015.

[79] NIST Group Big Data Public Working, "NIST Big Data Interoperability Framework: Volume 3, Use Cases and General Requirements," *NIST Spec. Publ.*, 2015.

[80] NIST Group Big Data Public Working, "NIST Big Data Interoperability Framework: Volume 7, Standards Roadmap," *NIST Spec. Publ.*, 2015.

[81] "Information technology Big data," *International Organization for Standardization*, 2014. [Online]. Available: https://www.iso.org/files/live/sites/isoorg/files/developing_standards/docs/en/big_data_report-jtc1.pdf. [Accessed: 24-May-2018].

[82] I. Gorton, J. Klein, and A. Nurgaliev, "Architecture Knowledge for Evaluating Scalable Databases," in *2015 12th Working IEEE/IFIP Conference on Software Architecture*, 2015, pp. 95–104.

[83] "QuABaseBD - Quality Architecture at Scale for Big Data." [Online]. Available: https://quabase.sei.cmu.edu/mediawiki/index.php/Main_Page. [Accessed: 03-Dec-2017].

[84] H.-M. Chen, R. Kazman, and S. Haziyev, "Strategic Prototyping for Developing Big Data Systems," *IEEE Softw.*, vol. 33, no. 2, pp. 36–43, Mar. 2016.

[85] H.-M. Chen, R. Kazman, and S. Haziyev, "Agile Big Data Analytics Development: An Architecture-Centric Approach," in *2016 49th Hawaii International Conference on System Sciences (HICSS)*, 2016, pp. 5378–5387.

[86] M. Maier, "Towards a Big Data Reference Architecture," University of Eindhoven, 2013.

[87] B. Geerdink, "A reference architecture for big data solutions introducing a model to perform predictive analytics using big data technology," in *8th International Conference for Internet Technology and Secured Transactions (ICITST-2013)*, 2013, pp. 71–76.

[88] P. Pääkkönen and D. Pakkala, "Reference Architecture and Classification of Technologies, Products and Services for Big Data Systems," *Big Data Res.*, vol. 2, no. 4, pp. 166–186, Feb. 2015.

[89] P. Viana and L. Sato, "A proposal for a reference architecture for long-term archiving, preservation, and retrieval of big data," in *Proceedings - 2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2014*, 2015, pp. 622–629.

[90] A. Cuzzocrea, “A Reference Architecture for Supporting Secure Big Data Analytics over Cloud-Enabled Relational Databases,” in *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, 2016, pp. 356–358.

[91] I. Polato, R. Ré, A. Goldman, and F. Kon, “A comprehensive view of Hadoop research - A systematic literature review,” vol. 46. Academic Press, pp. 1–25, 2014.

[92] A. T. Litchfield and J. Althouse, “A systematic review of cloud computing, big data and databases on the cloud,” in *AMCIS*, 2014.

[93] T. M. Abdellatif, L. F. Capretz, and D. Ho, “Software Analytics to Software Practice: A Systematic Literature Review,” in *2015 IEEE/ACM 1st International Workshop on Big Data Software Engineering*, 2015, pp. 30–36.

[94] X. Liu, N. Lftikhar, and X. Xie, “Survey of real-time processing systems for big data,” in *18th International Database Engineering & Applications Symposium*, 2014, pp. 356–361.

[95] M. Senthilkumar and P. Ilango, “A Survey on Job Scheduling in Big Data,” *Cybern. Inf. Technol.*, vol. 16, no. 3, Jan. 2016.

[96] J. V. Gautam, H. B. Prajapati, V. K. Dabhi, and S. Chaudhary, “A survey on job scheduling algorithms in Big data processing,” in *Electrical, Computer and Communication Technologies (ICECCT), 2015 IEEE International Conference*, 2015, pp. 1–11.

[97] P. Barlas, I. Lanning, and C. Heavey, “A survey of open source data science tools,” *Int. J. Intell. Comput. Cybern.*, vol. 8, no. 3, pp. 232–261, Aug. 2015.

[98] S. Tamboli and S. S. Patel, “A survey on innovative approach for improvement in efficiency of caching technique for big data application,” in *2015 International Conference on Pervasive Computing (ICPC)*, 2015, pp. 1–6.

[99] G. Caldiera, V. R. Basili, and H. D. Rombach, “The Goal Question Metric Approach,” *Encycl. Softw. Eng.*, vol. 2, pp. 528–532, 1994.

[100] Basili and V. R., “Software modeling and measurement: the Goal/Question/Metric paradigm.” University of Maryland at College Park, 1992.

[101] B. A. Kitchenham, D. Budgen, and O. Pearl Brereton, “Using mapping studies as the basis for further research – A participant-observer case study,” *Inf. Softw. Technol.*, vol. 53, no. 6, pp. 638–651, 2011.

[102] B. Kitchenham, “Guidelines for performing systematic literature reviews in software engineering,” in *Proceedings of the 28th international conference on Software engineering*, 2006, pp. 1051–1052.

[103] K. Petersen, S. Vakkalanka, and L. Kuzniarz, “Guidelines for conducting systematic mapping studies in software engineering: An update,” *Inf. Softw. Technol.*, vol. 64, pp. 1–18, 2015.

[104] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, “Systematic Mapping Studies in Software Engineering,” in *EASE*, 2008, pp. 68–77.

[105] P. Shanmugapriya and R. M. Suresh, “Software Architecture Evaluation Methods – A survey,” *Int. J. Comput. Appl.*, vol. 49, no. 16, pp. 975–8887, 2012.

[106] U. Suman and A. Patidar, “A Survey on Software Architecture Evaluation Methods,” in *Computing for Sustainable Global Development (INDIACom), 2015 2nd International*

Conference on, 2015, pp. 967–972.

[107] A. Avritzer and E. J. Weyuker, "Metrics to Assess the Likelihood of Project Success Based on Architecture Reviews," *Empir. Softw. Eng.*, vol. 4, no. 3, pp. 199–215, 1999.

[108] R. H. Reussner, H. W. Schmidt, and I. H. Poernomo, "Reliability prediction for component-based software architectures," *J. Syst. Softw.*, vol. 66, no. 3, pp. 241–252, Jun. 2003.

[109] M. Ionita, D. Hammer, and H. Obbink, "Scenario-Based Software Architecture Evaluation Methods: An Overview," *ICSE/SARA*, 2002.

[110] R. Kazman, G. Abowd, L. Bass, and P. Clements, "Scenario-Based Analysis of Software Architecture," *IEEE Softw.*, vol. 13, no. 6, pp. 47–55, 1996.

[111] J. C. R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, "The Architecture Tradeoff Analysis Method," in *Engineering of Complex Computer Systems, 1998. ICECCS'98. Proceedings. Fourth IEEE International Conference on*, 1998, pp. 68–78.

[112] P. Bengtsson, N. Lassing, J. Bosch, and H. van Vliet, "Architecture-Level Modifiability Analysis," *J. Syst. Softw.*, vol. 69, no. 1–2, pp. 129–147, 2004.

[113] R. Kazman, M. Klein, and P. Clements, "ATAM: Method for Architecture Evaluation," in *Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst*, 2000.

[114] R. Kazman, L. Bass, G. Abowd, and M. Webb, "SAAM: a method for analyzing the properties of software architectures," in *Proceedings of 16th International Conference on Software Engineering*, pp. 81–90.

[115] P. Bengtsson, N. Lassing, J. Bosch, and H. van Vliet, "Architecture-Level Modifiability Analysis," *J. Syst. Softw.*, vol. 69, no. 1–2, pp. 129–147, Jan. 2004.

[116] P. Bengtsson and J. Bosch, "Architecture level prediction of software maintenance," in *Proceedings of the Third European Conference on Software Maintenance and Reengineering (Cat. No. PR00090)*, pp. 139–147.

[117] P. Bengtsson and J. Bosch, "Scenario-based software architecture reengineering," in *Proceedings. Fifth International Conference on Software Reuse (Cat. No.98TB100203)*, pp. 308–317.

[118] G. Molter, "Integrating SAAM in Domain-centric and Reuse-based Development Processes," in *Proceedings of the 2nd Nordic Workshop on Software Architecture, Ronneby*, 1999, pp. 1–10.

[119] K. Bergner, A. Rausch, M. Sihling, and T. Ternité, "DoSAM – Domain-Specific Software Architecture Comparison Model," Springer, Berlin, Heidelberg, 2005, pp. 4–20.

[120] "Unified Modeling Language (Version 2.5)," 2015. [Online]. Available: http://www.omg.org/spec/UML/2.5/. [Accessed: 03-Dec-2017].

[121] K. Petersen and C. Gencel, "Worldviews, research methods, and their relationship to validity in empirical software engineering research," in *Proceedings - Joint Conference of the 23rd International Workshop on Software Measurement and the 8th International Conference on Software Process and Product Measurement, IWSM-MENSURA 2013*, 2013.

[122] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Berlin, Heidelberg: Springer, 2012.

[123] Organisation internationale de normalisation, *Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models*. ISO/IEC 25010, 2011.

[124] A. AL-Badareen, M. Selamat, M. Jabar, J. Din, and S. Turaev, "Software Quality Models: A Comparative Study," in *International Conference on Software Engineering and Computer Systems*, 2011, pp. 46–55.

[125] J. P. Miguel, D. Mauricio, and G. Rodríguez, "A Review of Software Quality Models for the Evaluation of Software Products," *Int. J. Softw. Eng. Appl.*, vol. 5, no. 6, 2014.

[126] A. Diaz-Pace, H. Kim, L. Bass, P. Bianco, and F. Bachmann, "Integrating Quality-attribute Reasoning Frameworks in the ArchE Design Assistant," in *International Conference on the Quality of Software Architectures*, 2008, pp. 171–188.

[127] N. B. Harrison and P. Avgeriou, "How do architecture patterns and tactics interact? A model and annotation," *J. Syst. Softw.*, vol. 83, no. 10, pp. 1735–1758, Oct. 2010.

[128] R. Sumbaly, J. Kreps, and S. Shah, "The big data ecosystem at LinkedIn," in *Proceedings of the 2013 international conference on Management of data - SIGMOD '13*, 2013, p. 1125.

[129] J. Kreps, L. Corp, N. Narkhede, and J. Rao, "Kafka: a Distributed Messaging System for Log Processing," in *Proceedings of the NetDB*, 2011, pp. 1–7.

[130] G. Mishne, J. Dalton, Z. Li, A. Sharma, and J. Lin, "Fast Data in the Era of Big Data: Twitter's Real-Time Related Query Suggestion Architecture," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, 2012, pp. 1147–1158.

[131] G. Lee, J. Lin, C. Liu, A. Lorek, and D. Ryaboy, "The unified logging infrastructure for data analytics at Twitter," *Proc. VLDB Endow.*, vol. 5, no. 12, pp. 1771–1780, Aug. 2012.

[132] M. Busch, K. Gade, B. Larson, P. Lok, S. Luckenbill, and J. Lin, "Earlybird: Real-Time Search at Twitter," in *2012 IEEE 28th International Conference on Data Engineering*, 2012, pp. 1360–1369.

[133] A. Magdy and M. F. Mokbel, "Towards a Microblogs Data Management System (Invited Industrial Paper)," in *Mobile Data Management (MDM), 2015 16th IEEE International Conference on*, 2015, pp. 271–278.

[134] M. L. Abbott and M. T. Fisher, *The art of scalability : scalable web architecture, processes, and organizations for the modern enterprise*. Pearson Education, 2009.

[135] L. Wu *et al.*, "Avatara: OLAP for Web-scale Analytics Products," *Proc. VLDB Endow.*, vol. 5, no. 12, pp. 1874–1877, 2012.

[136] "Using set cover algorithm to optimize query latency for a large scale distributed graph | LinkedIn Engineering." [Online]. Available: https://engineering.linkedin.com/real-time-distributed-graph/using-set-cover-algorithm-optimize-query-latency-large-scale-distributed. [Accessed: 14-Jul-2018].

[137] "Migrating to Espresso | LinkedIn Engineering." [Online]. Available: https://engineering.linkedin.com/blog/2017/08/migrating-from-oracle-to-espresso. [Accessed: 14-Jul-2018].

[138] G. Lee, J. Lin, C. Liu, A. Lorek, and D. Ryaboy, "The unified logging infrastructure for

data analytics at Twitter," *Proc. VLDB Endow.*, vol. 5, no. 12, pp. 1771–1780, 2012.

[139] J. Lin and D. Ryaboy, "Scaling big data mining infrastructure," *ACM SIGKDD Explor. Newsl.*, vol. 14, no. 2, p. 6, Apr. 2013.

[140] A. Thusoo *et al.*, "Data warehousing and analytics infrastructure at facebook," *Proc. 2010 ACM SIGMOD Int. Conf. Manag. data*, pp. 1013–1020, 2010.

[141] X. Amatriain, "Big & Personal: data and models behind Netflix recommendations," in *Proceedings of the 2nd international workshop on big data, streams and heterogeneous source Mining: Algorithms, systems, programming models and applications*, 2013, pp. 1–6.

[142] N. Marz and J. Warren, *Big data : principles and best practices of scalable real-time data systems*. New York; Manning Publications Co., 2015.

[143] J. Kroß, A. Brunnert, C. Prehofer, T. A. Runkler, and H. Krcmar, "Stream processing on demand for lambda architectures," in *European Workshop on Performance Engineering*, 2015, vol. 9272, pp. 243–257.

[144] M. Hausenblas and N. Bijnens, "Lambda Architecture," 2015. [Online]. Available: http://lambda-architecture.net/. [Accessed: 29-Oct-2017].

[145] "Kappa Architecture - Where Every Thing Is A Stream." [Online]. Available: http://milinda.pathirage.org/kappa-architecture.com/. [Accessed: 29-Oct-2017].

[146] C. E. Cuesta, M. A. Martínez-Prieto, and J. D. Fernández, "Towards an architecture for managing big semantic data in real-time," in *European Conference on Software Architecture*, 2013, vol. 7957 LNCS, pp. 45–53.

[147] S. Angelov, P. Grefen, and D. Greefhorst, "A framework for analysis and design of software reference architectures," *Inf. Softw. Technol.*, vol. 54, no. 4, pp. 417–431, Apr. 2012.

[148] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS Q.*, vol. 28, no. 1, pp. 75–105, 2004.

[149] D. Huhnlein, U. Korte, L. Langer, and A. Wiesmaier, "A Comprehensive Reference Architecture for Trustworthy Long-Term Archiving of Sensitive Data," in *2009 3rd International Conference on New Technologies, Mobility and Security*, 2009, pp. 1–5.

[150] M. Butler, D. Reynolds, I. Dickinson, B. McBride, D. Grosvenor, and A. Seaborne, "Semantic Middleware for E-Discovery," in *2009 IEEE International Conference on Semantic Computing*, 2009, pp. 275–280.

[151] EDRM, "The electronic discovery reference model." [Online]. Available: http://www.edrm.net/resources/guides/edrm-framework-guides. [Accessed: 03-Mar-2018].

[152] F. Buschmann, K. Henney, and D. C. Schmidt, *Pattern-oriented software architecture. v. 4, A pattern language for distributed computing*. John Wiley, 2007.

[153] "Data Architecture Lessons from LinkedIn.com Data Handling Mechanisms - Reskilling IT." [Online]. Available: https://vitalflux.com/data-handled-linkedin-com/. [Accessed: 25-Nov-2017].

[154] "A Brief History of Scaling LinkedIn | LinkedIn Engineering." [Online]. Available: https://engineering.linkedin.com/architecture/brief-history-scaling-linkedin. [Accessed: 18-Jul-2018].

[155] “Introducing Espresso - LinkedIn’s hot new distributed document store | LinkedIn Engineering.” [Online]. Available: https://engineering.linkedin.com/espresso/introducing-espresso-linkedins-hot-new-distributed-document-store. [Accessed: 18-Jul-2018].

[156] “Introducing and Open Sourcing Ambry - LinkedIn’s New Distributed Object Store | LinkedIn Engineering.” [Online]. Available: https://engineering.linkedin.com/blog/2016/05/introducing-and-open-sourcing-ambry---linkedins-new-distributed-. [Accessed: 18-Jul-2018].

[157] M. Slee, A. Agarwal, and M. Kwiatkowski, “Thrift: Scalable Cross-Language Services Implementation,” *Faceb. White Pap.*, vol. 5, no. 8, 2007.

[158] G. Kaur and M. M. Fuad, “An evaluation of Protocol Buffer,” in *Proceedings of the IEEE SoutheastCon 2010*, 2010, pp. 459–462.

[159] “The Infrastructure Behind Twitter: Scale.” [Online]. Available: https://blog.twitter.com/engineering/en_us/topics/infrastructure/2017/the-infrastructure-behind-twitter-scale.html. [Accessed: 18-Jul-2018].

[160] “Introducing Gizzard, a framework for creating distributed datastores.” [Online]. Available: https://blog.twitter.com/engineering/en_us/a/2010/introducing-gizzard-a-framework-for-creating-distributed-datastores.html. [Accessed: 18-Jul-2018].

[161] “Introducing FlockDB.” [Online]. Available: https://blog.twitter.com/2010/introducing-flockdb. [Accessed: 18-Jul-2018].

[162] “Blobstore: Twitter’s in-house photo storage system.” [Online]. Available: https://blog.twitter.com/engineering/en_us/a/2012/blobstore-twitter-s-in-house-photo-storage-system.html. [Accessed: 18-Jul-2018].

[163] “Manhattan, our real-time, multi-tenant distributed database for Twitter scale.” [Online]. Available: https://blog.twitter.com/engineering/en_us/a/2014/manhattan-our-real-time-multi-tenant-distributed-database-for-twitter-scale.html. [Accessed: 18-Jul-2018].

[164] M. Dusi *et al.*, “Blockmon: Flexible and High-Performance Big Data Stream Analytics Platform and its Use Cases,” *NEC Tech. J.*, vol. 7, no. 2, p. 103, 2012.

[165] D. Abadi, “Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story,” *Computer (Long. Beach. Calif).*, vol. 45, no. 2, pp. 37–42, Feb. 2012.

[166] M. Stonebraker, “The Case for Shared Nothing,” *Database Eng.*, vol. 9, no. 1, pp. 4–9, 1986.

[167] T. Müseler, “A survey of Shared-Nothing Parallel Database Management Systems [Comparison between Teradata, Greenplum and Netezza implementations],” *IRCSE*, 2012.

[168] “You Can’t Sacrifice Partition Tolerance | codahale.com.” [Online]. Available: https://codahale.com/you-cant-sacrifice-partition-tolerance/. [Accessed: 19-Nov-2017].

[169] D. B. Terry, V. Prabhakaran, R. Kotla, M. Balakrishnan, M. K. Aguilera, and H. Abu-Libdeh, “Consistency-based service level agreements for cloud storage,” in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles - SOSP ’13*, 2013, pp. 309–324.

[170] G. Deepak, “A Critical Comparison of NOSQL Databases in the Context of Acid and

Base," *IRCSE*, 2016.

[171] T. Cerqueus, E. C. de Almeida, and S. Scherzinger, "Safely Managing Data Variety in Big Data Software Development," in *2015 IEEE/ACM 1st International Workshop on Big Data Software Engineering*, 2015, pp. 4–10.

[172] M. Klettke, U. Storl, M. Shenavai, and S. Scherzinger, "NoSQL schema evolution and big data migration at scale," in *2016 IEEE International Conference on Big Data (Big Data)*, 2016, pp. 2764–2774.

[173] I. Labs, T. Dumitras¸and, D. Dumitras¸and, and M. Hicks, "Evolving NoSQL Databases Without Downtime," in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2016, pp. 166–176.

[174] D. Ganesh Chandra, "BASE analysis of NoSQL database," *Futur. Gener. Comput. Syst.*, vol. 52, pp. 13–21, Nov. 2015.

[175] F. Gessert, W. Wingerath, S. Friedrich, and N. Ritter, "NoSQL database systems: a survey and decision guidance," *Comput. Sci. - Res. Dev.*, vol. 32, no. 3–4, pp. 353–365, Jul. 2017.

[176] G. C. Deka, "NoSQL Polyglot Persistence," in *Advances in Computers*, Elsevier, 2018, pp. 357–390.

[177] E. Płuciennik and K. Zgorzałek, "The Multi-model Databases – A Review," Springer, Cham, 2017, pp. 141–152.

[178] J. Kreps, *I love logs : event data, stream processing, and data integration*. O'Reilly Media, 2015.

[179] A. D. Birrell and B. J. Nelson, "Implementing remote procedure calls," *ACM Trans. Comput. Syst.*, vol. 2, no. 1, pp. 39–59, Feb. 1984.

[180] M. Kleppmann, *Designing data-intensive applications : the big ideas behind reliable, scalable, and maintainable systems*. O'Reilly Media, Inc., 2017.

[181] J. Pérez-Martínez and A. Sierra, "A Taxonomy of the Quality Attributes for Distributed Applications." Informatica, 2002.

[182] K. V. Vishwanath and N. Nagappan, "Characterizing cloud computing hardware reliability," in *Proceedings of the 1st ACM symposium on Cloud computing - SoCC '10*, 2010, p. 193.

[183] J. P. Russo and M. Solari, "Estudio de Mapeo Sistemático sobre Arquitecturas de Software para Big Data," in *XX Ibero-American Conference on Software Engineering (CIbSE)*, 2017, pp. 747–759.

[184] R. Kitchin and G. Mcardle, "The diverse nature of big data," *SSRN Electron. J.*, 2015.

[185] S. Wang, S. Schlobach, and M. Klein, "What Is Concept Drift and How to Measure It?," Springer Berlin Heidelberg, 2010, pp. 241–256.

[186] P. Zikopoulos, D. DeRoos, K. Parasuraman, T. Deutsch, J. Giles, and D. Corrigan, *Harness the Power of Big Data: The IBM Big Data Platform*. 2013.

[187] C. Wu, R. Buyya, and K. Ramamohanarao, "Big Data Analytics = Machine Learning + Cloud Computing," *arXiv*, Jan. 2016.