Analytic Distributed DBMS(ADDB) 소개

연세대학교 컴퓨터과학과 JINHUIJUN

2022년 08월

과제명: IoT 환경을 위한 고성능 플래시 메모리 스토리지 기반 인메모리 분산 DBMS 연구개발

과제번호: 2017-0-00477

과학기술정보통신부
Ministry of Science and ICT

연세대학교
YONSEI UNIVERSITY

정보통신기술진흥센터
Institute for Information & communications Technology Promotion

01 과제 개요

과제 개요

과제의 목표 및 필요성

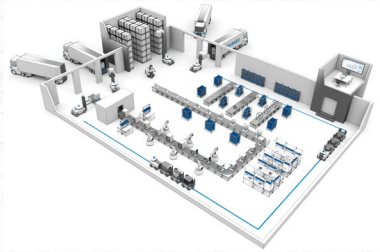
과제의 목표 :

엑사스케일(**Exa-scale**) 규모의 데이터를 저장하고,
초당 기가 스케일(**Giga-scale**)의 트랜잭션을 처리할 수 있는
DRAM 및 플래시 메모리 스토리지의 계층 구조 기반
분산 **DBMS** 연구 개발 → **IoT** 데이터를 중심으로

가트너에 따르면 2017년까지 IoT 디바이스가 2016년 대비 31% 증가한 84억개가 될 것이며, 2020년까지 204억개에 달할 것으로 예상됨

"Forecast: Internet of Things — Endpoints and Associated Services, Worldwide, 2016", Gartner

Connected Car (Components)

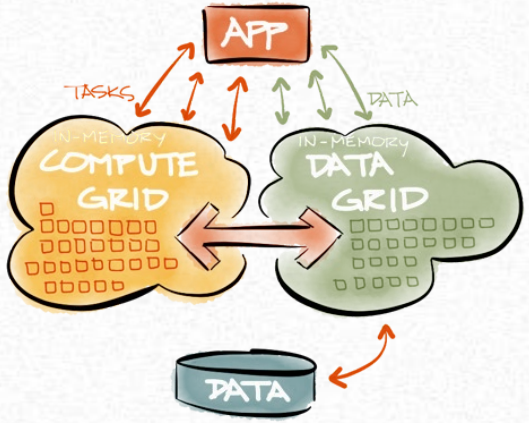


Smart Factory (Sensor devices)

Smart Home (Home appliances)

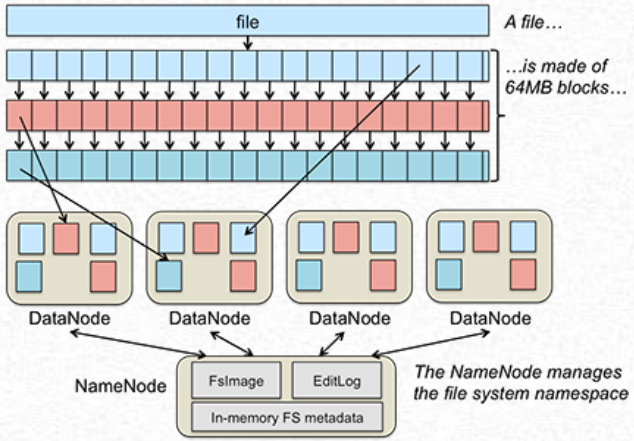
과제 개요

기존 시스템의 한계 및 문제점



In-memory Data Grid
(e.g. Oracle Coherence)

- ✓ 모든 데이터가 memory에 존재해 매우 빠른 처리가 가능
- ✓ 가격 대비 용량 면에서 대용량 데이터 적재 시 비효율적



HDFS(Hadoop File System)

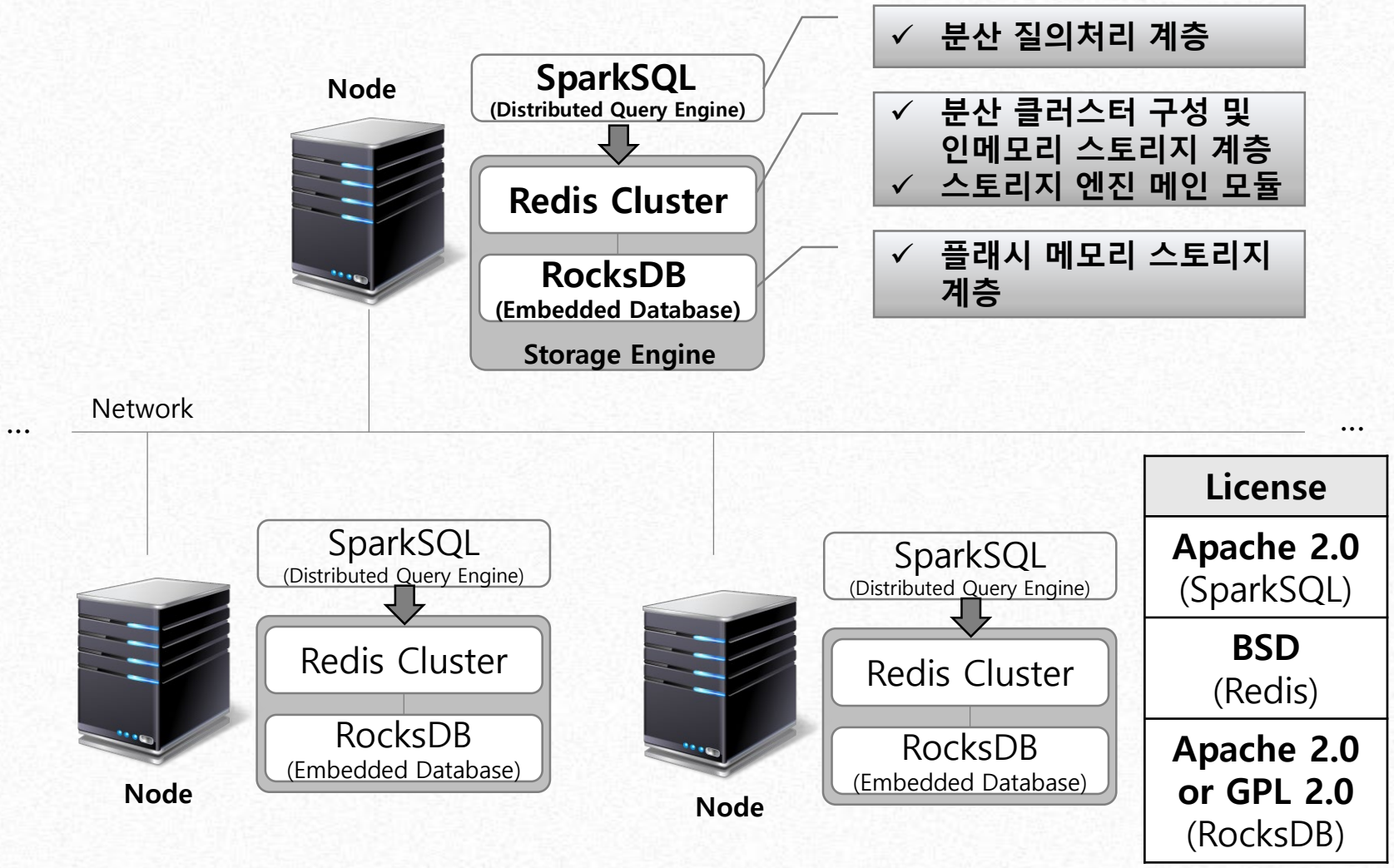
- ✓ HDD에 최적화된 대용량 분산 데이터 저장소
- ✓ IoT 환경의 small-sized 데이터 처리 시 과도한 I/O로 느린 성능

기존 시스템의 한계를 극복하기 위하여 small-sized 데이터 처리에 적합한 인메모리 및 디스크 기반 NoSQL 데이터베이스를 결합 →
인메모리 데이터베이스에 가까운 성능을 내면서 대용량 데이터 저장이 용이해 짐

02 연구 최종 목표

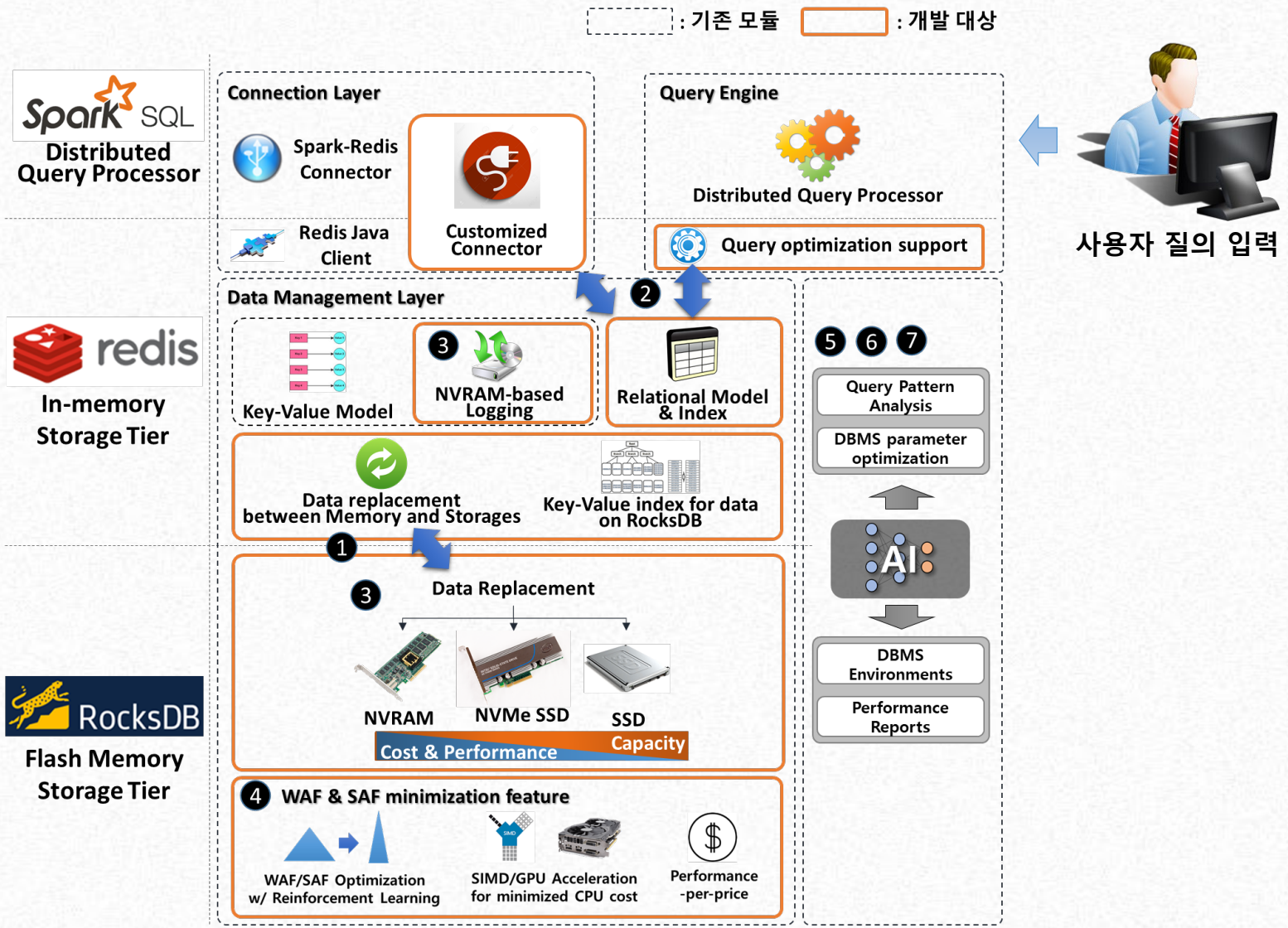
연구 목표

기술개발 대상 시스템 구성도 – 기존 공개SW를 활용한 시스템 구성



연구목표

기술개발 대상 시스템 세부 모듈



Apache Spark

연세대학교 컴퓨터과학과 박상현
2018년 8월

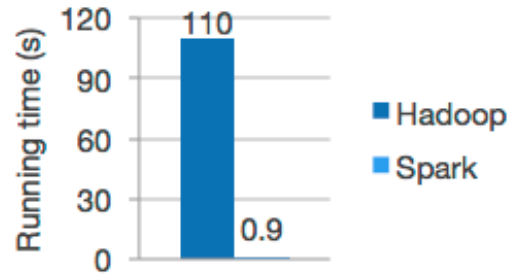
Apache Spark

대용량 데이터 처리/분석을 위한 "범용 분산 클러스터링 연산 프레임워크"

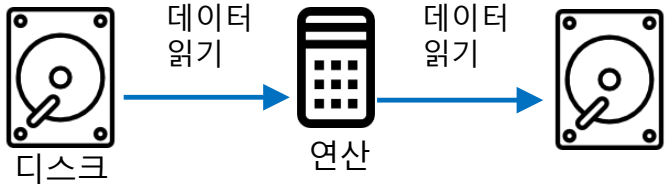
- 2009년 UC Berkely AMPLab에서 시작된 오픈소스 프로젝트
- 2014년 아파치 프로젝트 선정
- 2014년 아파치 최상위 프로젝트 선정
- 2018년 Stable release 2.3.1 (Licensed under the Apache License, Version 2.0.)
- 구현 언어 - Scala(함수형 언어)

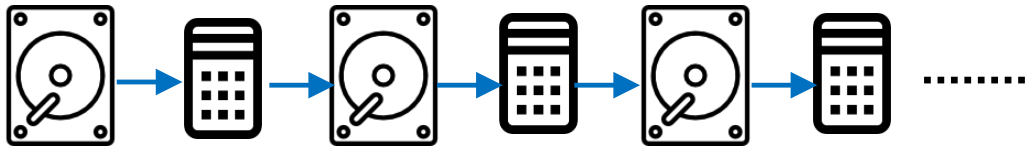
Why Apache Spark?

1. **성능** - 인메모리 기반의 Map&Reduce 연산을 통해 하둡의 MR보다 빠른 성능 구현

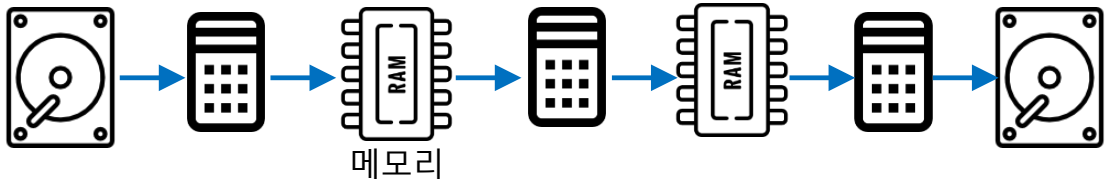


Logistic regression in Hadoop and Spark





반복 연산일 때 효율이 감소



중간 결과를 **메모리** 상에 유지할 수 있다면,
매번 디스크에 쓰는 것 보다 훨씬 빠르다

그림 참고 https://www.youtube.com/watch?v=DnLVkVQk73E 27:40

Why Apache Spark?

2. **다수의 언어 인터페이스 지원** - Java, **Scala**, Python, R

특히 Scala를 사용한 함수형 언어가 데이터 변환에 효과적

➔ 간결한 구현이 가능

Why Apache Spark?

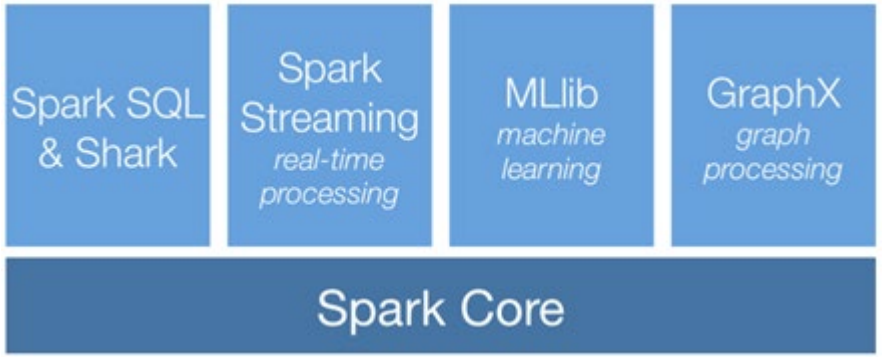
3. **범용성** - 단일 프로그램 내에서 여러 어플리케이션 적용 가능

: 실시간 데이터 처리 및 분석,

SQL를 통한 데이터 분석,

Machine learning,

Graph 분석

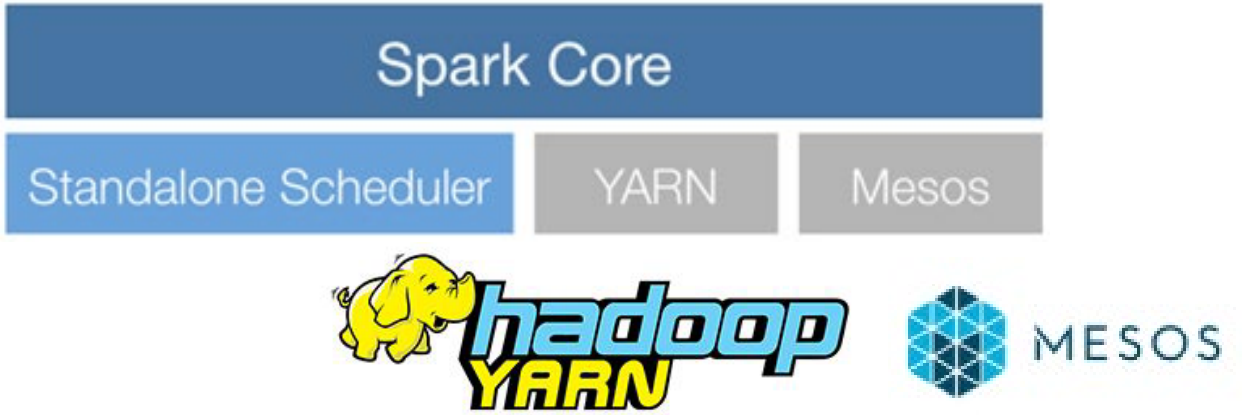


Why Apache Spark?

4. **데이터 활용성** - HDFS, Apache HBase, Apache Cassandra 등 다양한 데이터 소스에 사용 가능

Why Apache Spark?

5. **클러스터링 리소스 관리** - YARN, Apache Mesos 등 클러스터링 리소스 관리 패키지를 통해 클러스터링 환경에서의 동작 최적화

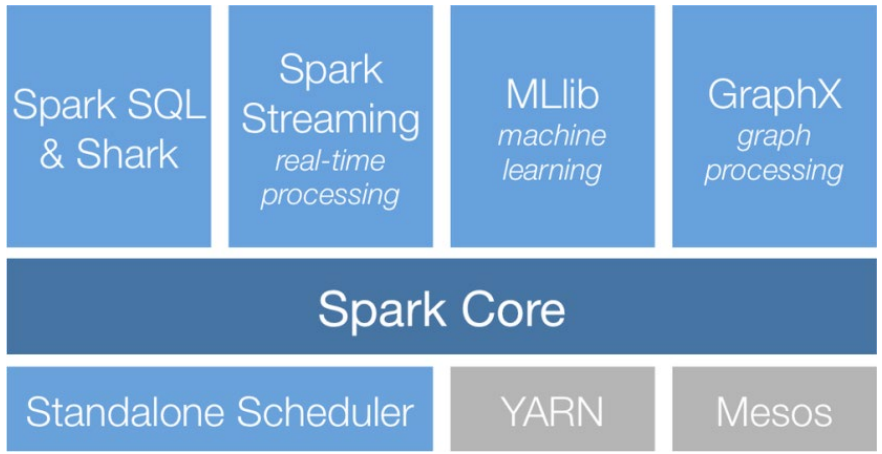


Master - Slave 관계를 통한 Resource 관리

vs. Apache Hadoop

	Apache Hadoop	Apache Spark
사용 목적	분산 데이터 스토리지 및 데이터 처리와 분석	빠른 데이터 처리와 분석, 손쉬운 관리와 범용성
데이터 스토리지	포함 (HDFS)	미포함 (그러나 HDFS, Cassandra, HBase 등 사용 가능)
프로젝트 관리	복잡 (많은 툴 숙지 필요)	쉬움
성능	느림	빠름
데이터 처리 기반 하드웨어	디스크	메모리
복구	지원	지원

Architecture



- **SparkSQL** : 데이터에 대한 SQL 처리
- **Spark Streaming** : 데이터 스트리밍 처리
- **MLlib** : Machine learning
- **GraphX** : Vertices + Edges로 이루어진 Graph 분석

- **Spark Core** : 메인 컴포넌트이며, 전체적인 프로젝트의 기반. Task dispatching, scheduling, **RDD** 기반의 연산 등 데이터 처리를 위한 작업을 수행

- Standalone Scheduler : 싱글 노드 환경 구축
- **YARN, Mesos** : 클러스터 리소스 관리 및 환경 구축

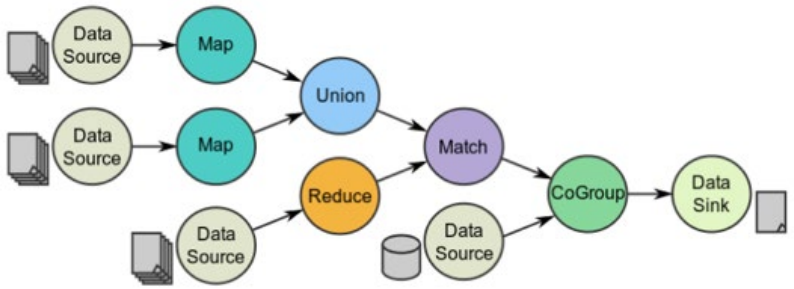
RDD (1/2)

- Spark의 핵심 자료구조. Spark 내의 모든 데이터는 RDD 타입으로 처리
- RDD := Resilient Distributed Dataset - 탄력적 분산 데이터셋
  - **Immutable**, **partitioned** collections of record
  - 인메모리 기반으로 작동하는 Spark를 위해, 복구 기능을 포함한 가상의 자료구조

1) **In-Memory** 데이터 처리 - 디스크 기반보다 매우 빠른 속도, 하지만 Fault 복구 문제

2) **Fault-tolerant** - Update를 없애고 메모리에 read-only로만 쓰기 → "**immutable**"

Lineage(계보)를 두어 RDD에 적용된 순서를 모두 저장 → 복구&연산 용이



[Lineage 예시]

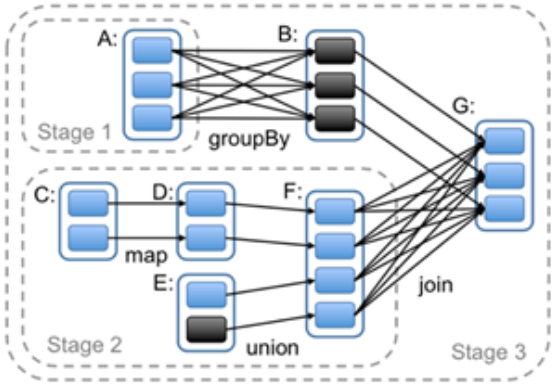
RDD (2/2)

- Spark의 핵심 자료구조. Spark 내의 모든 데이터는 RDD 타입으로 처리
- RDD := Resilient Distributed Dataset - 탄력적 분산 데이터셋

3) **Lazy-execution** - 실제 RDD 연산은 데이터를 생성할 때(Action operation)만 Lineage를 통해 최적 수행

4) "**Partitioned**" - 데이터를 **파티션**으로 나누어 클러스터 노드에 분산 처리함

5) **Job scheduling** - Lineage를 바탕으로 데이터의 배치와 파티션 분배를 미리 계산



[Partition, Job scheduling 예시]

RDD basic operations

* RDD operations
  1) **Transformation** - RDD를 변환하여 새로운 RDD 리턴
  2) **Action** - RDD를 이용하여 최종 계산 결과 리턴
- Spark은 **Lazy execution**을 사용하여,
  Transformation을 lineage에 기록하다가 Action이 실행될 때 실제 연산을 수행

Transformations	*map*(*f* : T ⇒ U)	:	RDD[T] ⇒ RDD[U]
	filter(*f* : T ⇒ Bool)	:	RDD[T] ⇒ RDD[T]
	flatMap(*f* : T ⇒ Seq[U])	:	RDD[T] ⇒ RDD[U]
	sample(*fraction* : Float)	:	RDD[T] ⇒ RDD[T] (Deterministic sampling)
	groupByKey()	:	RDD[(K, V)] ⇒ RDD[(K, Seq[V])]
	reduceByKey(*f* : (V, V) ⇒ V)	:	RDD[(K, V)] ⇒ RDD[(K, V)]
	union()	:	(RDD[T], RDD[T]) ⇒ RDD[T]
	join()	:	(RDD[(K, V)], RDD[(K, W)]) ⇒ RDD[(K, (V, W))]
	cogroup()	:	(RDD[(K, V)], RDD[(K, W)]) ⇒ RDD[(K, (Seq[V], Seq[W]))]
	crossProduct()	:	(RDD[T], RDD[U]) ⇒ RDD[(T, U)]
	mapValues(*f* : V ⇒ W)	:	RDD[(K, V)] ⇒ RDD[(K, W)] (Preserves partitioning)
	sort(*c* : Comparator[K])	:	RDD[(K, V)] ⇒ RDD[(K, V)]
	partitionBy(*p* : Partitioner[K])	:	RDD[(K, V)] ⇒ RDD[(K, V)]
Actions	*count*()	:	RDD[T] ⇒ Long
	collect()	:	RDD[T] ⇒ Seq[T]
	reduce(*f* : (T, T) ⇒ T)	:	RDD[T] ⇒ T
	lookup(*k* : K)	:	RDD[(K, V)] ⇒ Seq[V] (On hash/range partitioned RDDs)
	save(*path* : String)	:	Outputs RDD to a storage system, *e.g.*, HDFS

그림: http://pubdata.tistory.com/38
공식 문서: https://spark.apache.org/docs/latest/rdd-programming-guide.html

Data Analysis Workflow

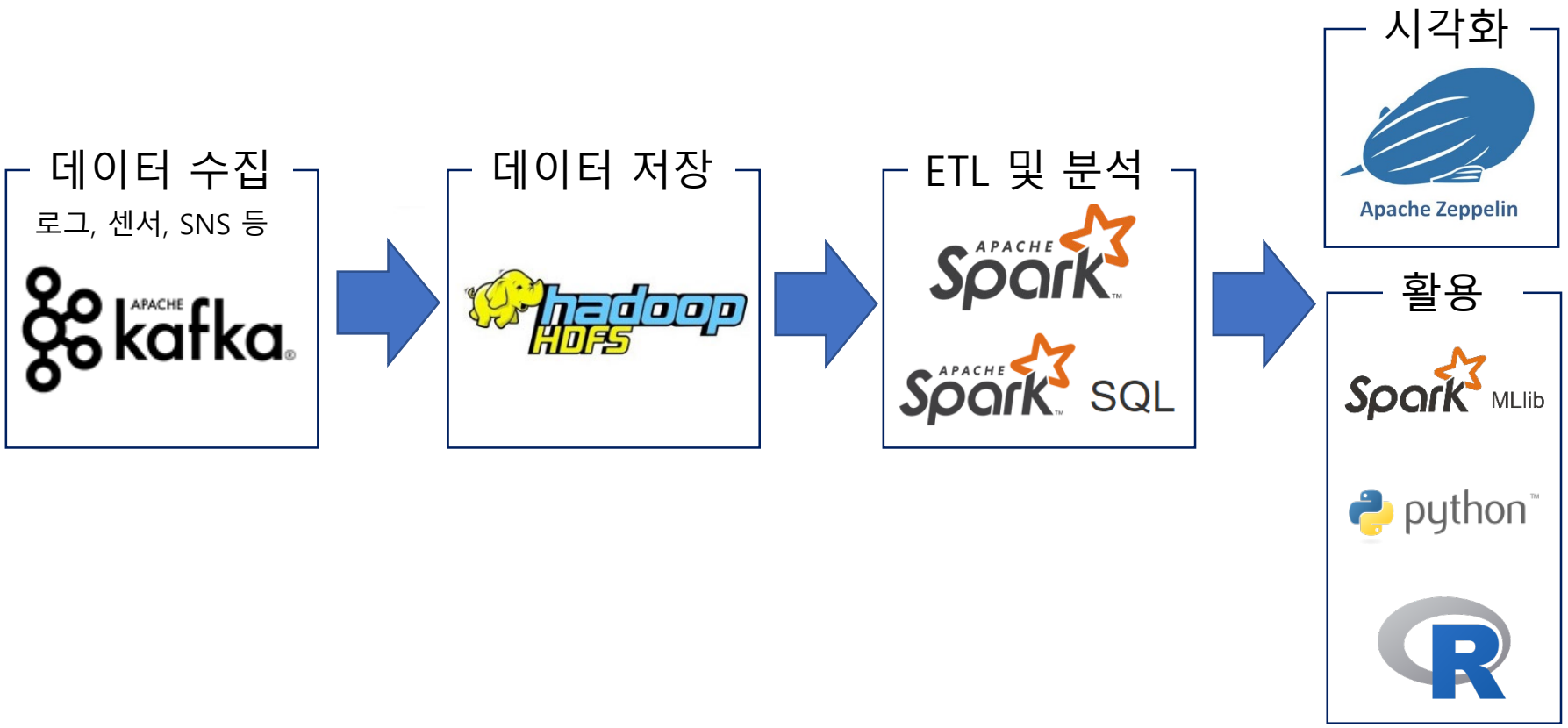


그림 참고 https://www.youtube.com/watch?v=DnLVkVQk73E 47:00

Application

- 대용량 데이터를 분산 처리하기 위해 기존에 사용했던 Apache Hadoop을 대체 (기반으로 사용)
- 기능적 활용성 - 다양한 어플리케이션 적용 가능(SQL, Streaming, ML, Graph)
- Spark의 발전 가능성

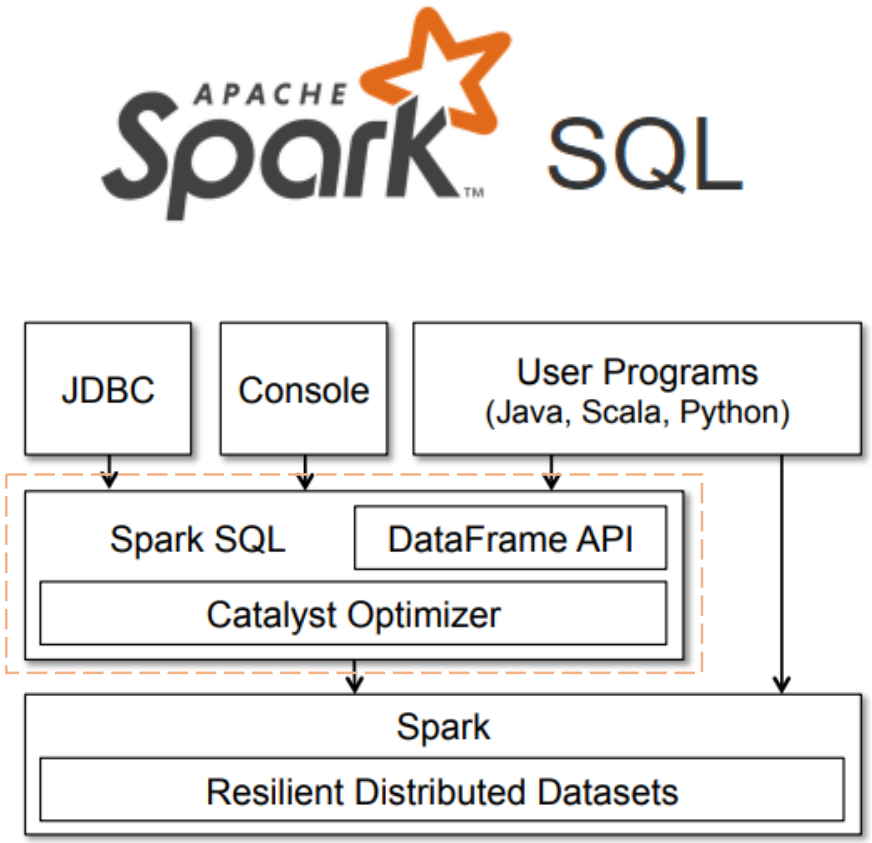
  현재 빅데이터의 분석,처리용으로 이미 수많은 기업에서 사용되고 있음

  좋은 성능과 인터페이스, 확장성 + 수많은 사용자와 개발자 ➔ 계속 발전할 것으로 전망

SparkSQL

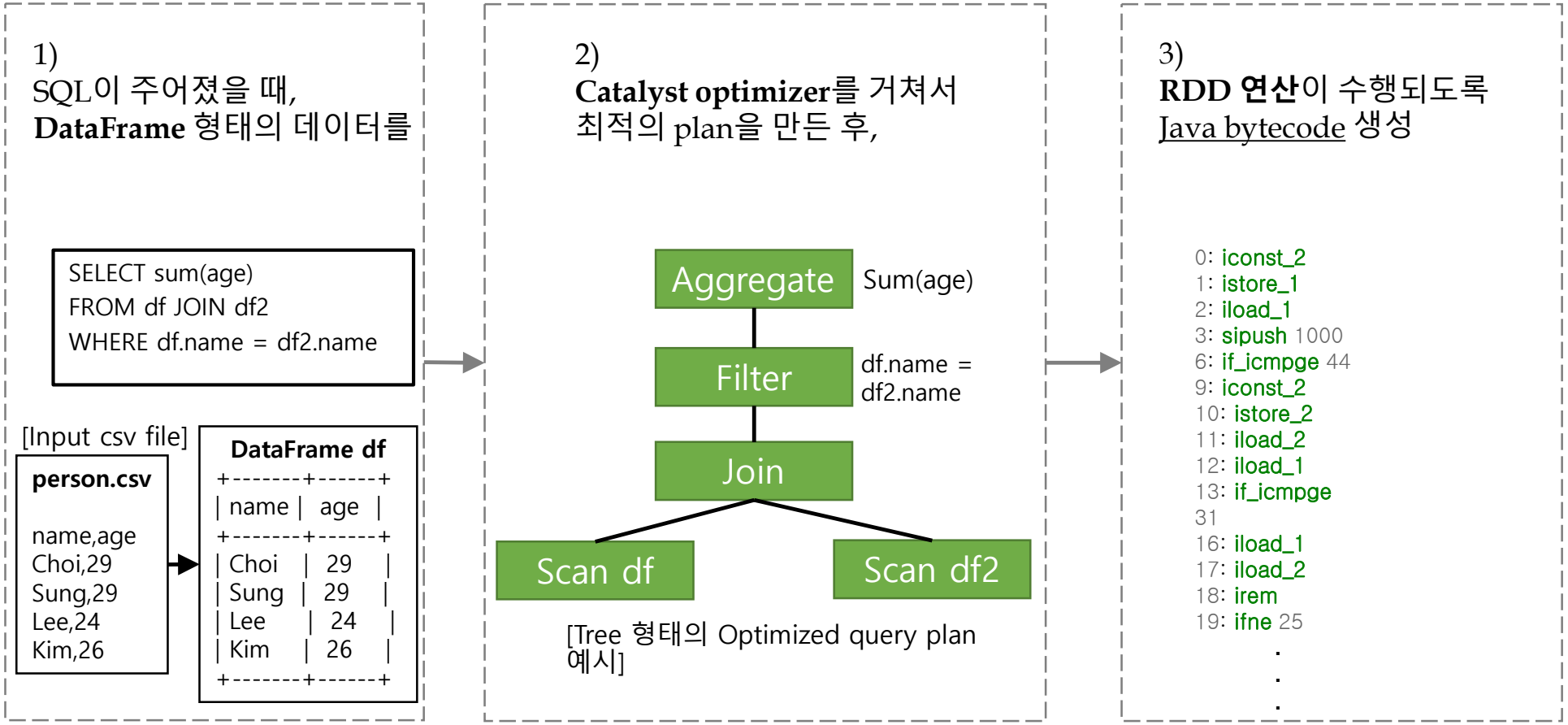
연세대학교 컴퓨터과학과 박상현
2018년 8월

SparkSQL



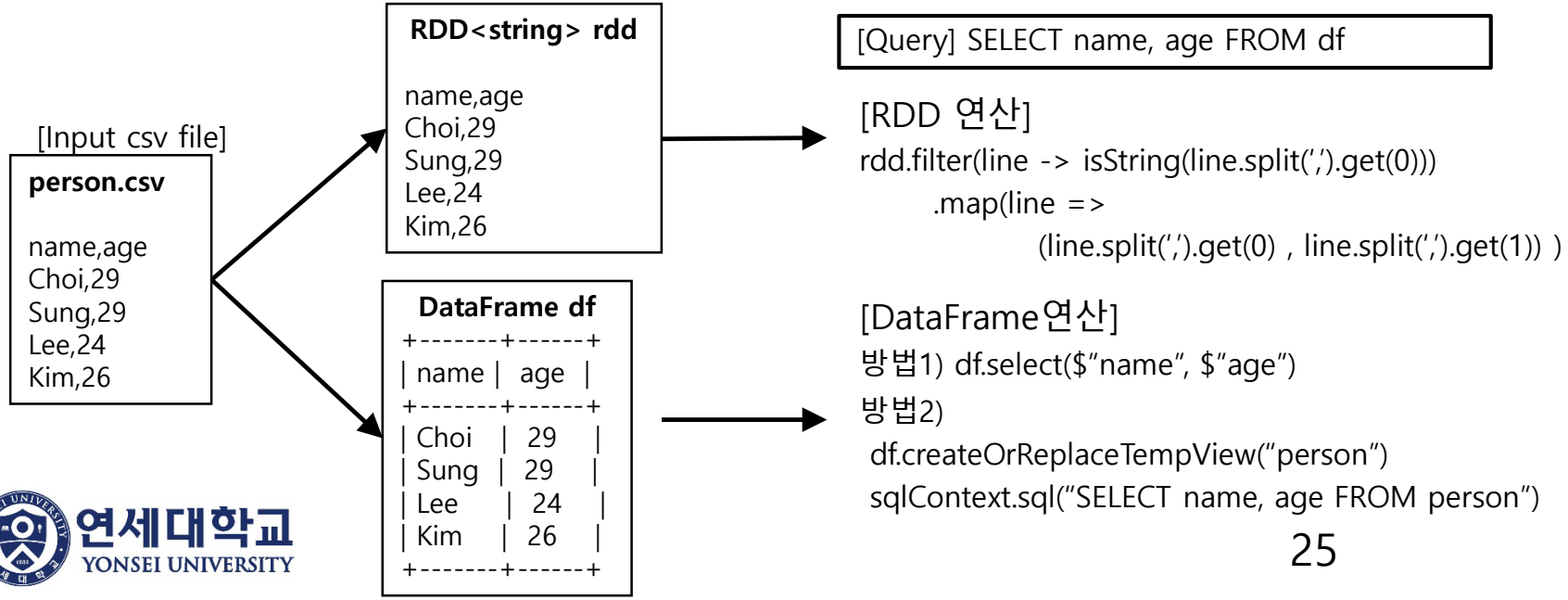
- Spark에서 대용량 데이터를 다룰 때, 관계형 데이터베이스에서 사용하는 SQL statement를 적용할 수 있도록 만든 라이브러리
  → 관계형 데이터베이스가 보편화 되어있기 때문
- 주 요소로 **DataFrame API**와 **Catalyst Optimizer**를 사용
- Spark에서 사용하는 RDD를 SQL로 적절하게 다룰 수 있어야함을 고려
  Procedural(RDD) → **Declarative**(SQL)
- JDBC/ODBC 와 같은 표준적인 커넥션을 제공함

Query execution flow



Dataframe API

- DataFrame = RDD + Table schema
- Spark의 장점을 살리기 위해선 RDD를 꼭 사용해야 하고, SQL을 적용하기 위해서는 table의 schema 정보를 포함하는 형태가 필요함 ➔ DataFrame
- DataFrame은 관계형 데이터베이스에서 Table 혹은 Relation으로 생각할 수 있음
- 관계형 Operation을 사용할 수 있으며, 이에 따른 Optimizer를 적용할 수 있음



Enhancement

- 지속적으로 SQL의 기능 개발과 성능 개선을 진행중 (~v2.3.1)

  Shark - SparkSQL 이전의 Spark 기반 SQL 모델

  → SparkSQL - DataFrame API, Optimizer

  → DataSet - RDD와 DataFrame 각각의 장점만을 취하는 새로운 구조

  → 그 밖의 성능적 개선이 계속적으로 진행 - Data source API 성능 개선 등

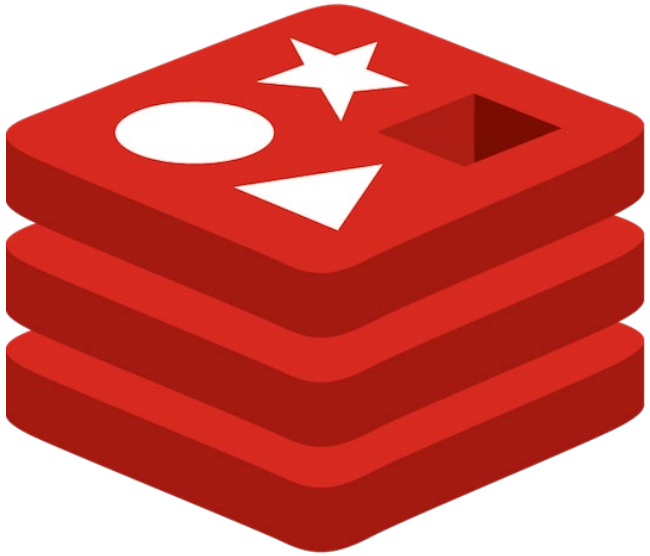
Redis

REmote **DI**ctionary **S**erver

연세대학교 컴퓨터과학과 박상현
2018년 8월

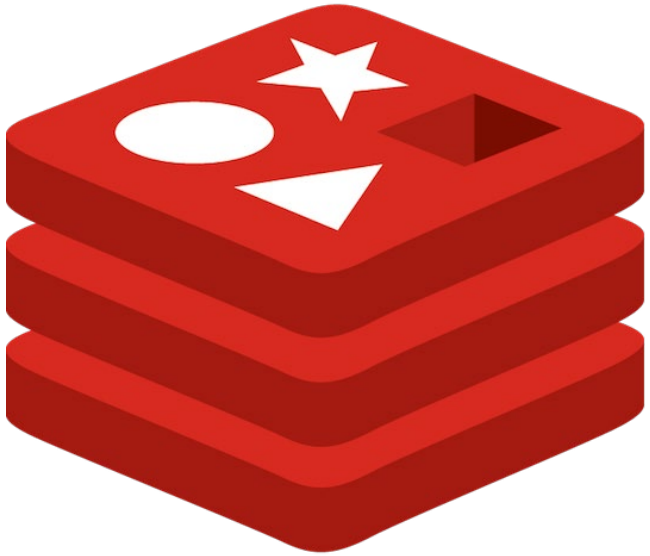
연세대학교
YONSEI UNIVERSITY

Redis?



- In-memory Key-Value Database
  - 메모리에 Key-Value pair로 데이터를 저장함
  - 다양한 데이터 타입 지원
    - String
    - Hash
    - List
    - Set
    - …
  - Implemented by “C”
  - Single-Thread 기반
  - Light weight

Redis?



- Advance Functions
  - Clustering을 통하여 scalable하게 구성 가능
  - Transaction을 통하여 atomic operation 제공
  - Caching & Message Queue에 특화된 Utility-tool 제공

Redis History

- Salvatore Sanfilippo(a.k.a. antirez)
  Redis Labs가 주도적으로 개발

- **2009**
  Initial release (May 10)
  Stablized release
- **2010**
  VMWare hired "antirez" (March)
- **2013**
  Sponsored by "Pivotal Software"
  → VMware spin-off
- **2015**
  Sponsored by "Redis Labs"
- **2018**
  Release "4.0.9" (March 27)

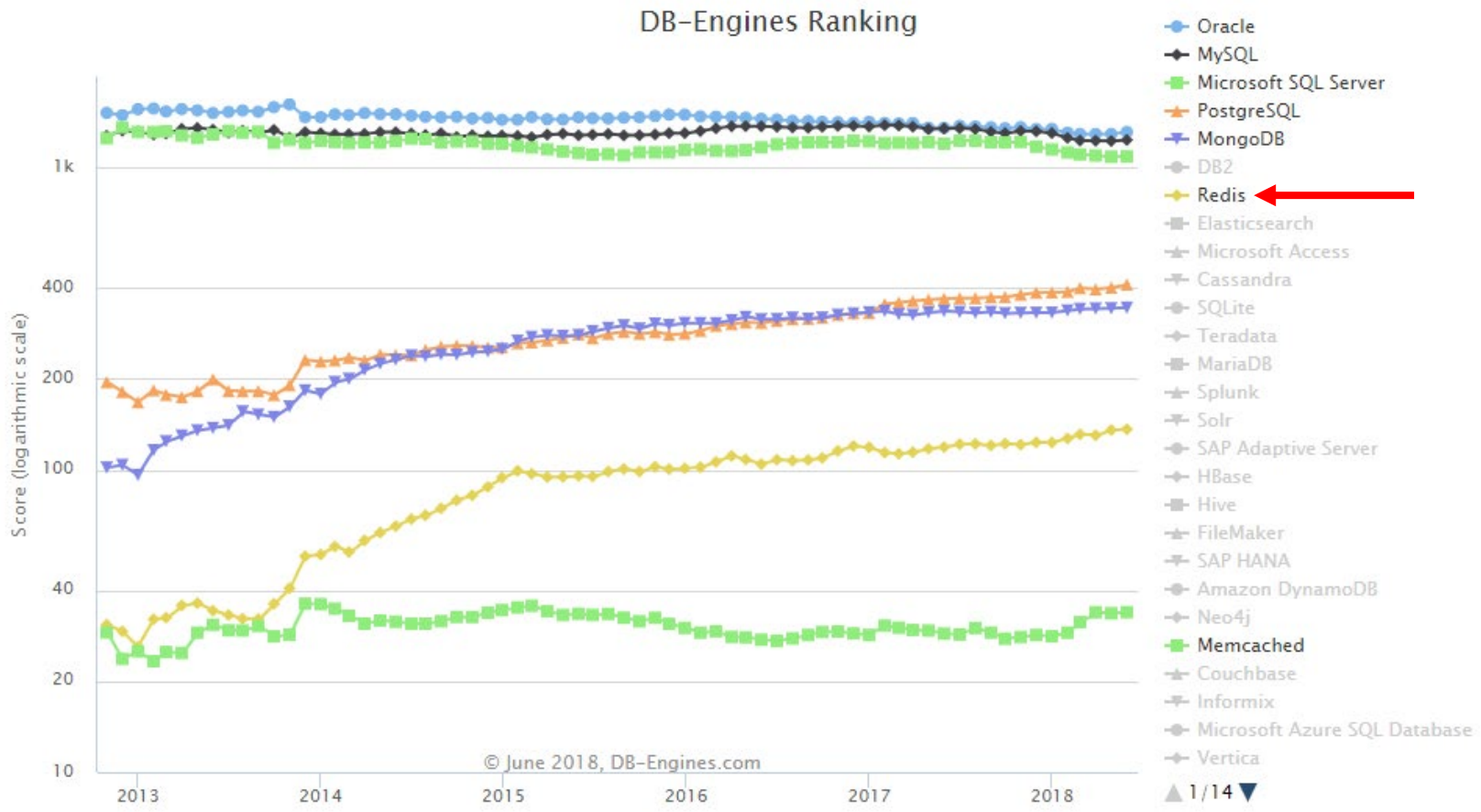
Database Market Share

343 systems in ranking, June 2018

Rank Jun 2018	Rank May 2018	Rank Jun 2017	DBMS	Database Model	Score Jun 2018	Score May 2018	Score Jun 2017
1.	1.	1.	Oracle	Relational DBMS	1311.25	+20.84	-40.51
2.	2.	2.	MySQL	Relational DBMS	1233.69	+10.35	-111.62
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1087.73	+1.89	-111.23
4.	4.	4.	PostgreSQL	Relational DBMS	410.67	+9.77	+42.13
5.	5.	5.	MongoDB	Document store	343.79	+1.67	+8.79
6.	6.	6.	DB2	Relational DBMS	185.64	+0.03	-1.86
7.	7.	↑ 9.	Redis	Key-value store	136.30	+0.95	+17.42
8.	↑ 9.	↑ 11.	Elasticsearch	Search engine	131.04	+0.60	+19.48
9.	↓ 8.	↓ 7.	Microsoft Access	Relational DBMS	130.99	-2.12	+4.44
10.	10.	↓ 8.	Cassandra	Wide column store	119.21	+1.38	-4.91
11.	11.	↓ 10.	SQLite	Relational DBMS	114.26	-1.19	-2.44
12.	12.	12.	Teradata	Relational DBMS	75.77	+1.36	-1.55
13.	↑ 14.	↑ 18.	MariaDB	Relational DBMS	65.85	+0.85	+12.95
14.	↓ 13.	↑ 16.	Splunk	Search engine	65.78	+0.68	+8.26
15.	15.	↓ 14.	Solr	Search engine	62.06	+0.55	-1.55
16.	16.	↓ 13.	SAP Adaptive Server	Relational DBMS	61.49	-0.02	-6.04
17.	17.	↓ 15.	HBase	Wide column store	59.70	-0.25	-2.17
18.	18.	↑ 20.	Hive	Relational DBMS	57.33	+0.36	+12.95
19.	19.	↓ 17.	FileMaker	Relational DBMS	56.18	+1.51	-0.90
20.	20.	↓ 19.	SAP HANA	Relational DBMS	49.35	+0.97	+1.85
21.	21.	↑ 22.	Amazon DynamoDB	Multi-model	45.79	+1.60	+11.78
22.	22.	↓ 21.	Neo4j	Graph DBMS	41.97	+1.39	+4.10
23.	23.	↑ 24.	Memcached	Key-value store	33.80	+0.25	+5.07
24.	24.	↓ 23.	Couchbase	Document store	32.45	+0.04	+0.54

Ranked by DB-Engines
https://db-engines.com/en/ranking

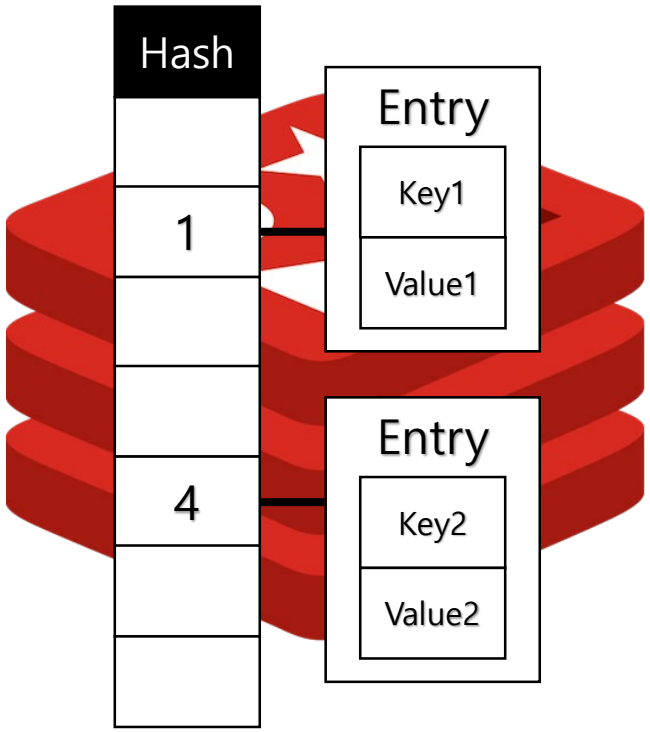
Database Market Share



연세대학교 YONSEI UNIVERSITY

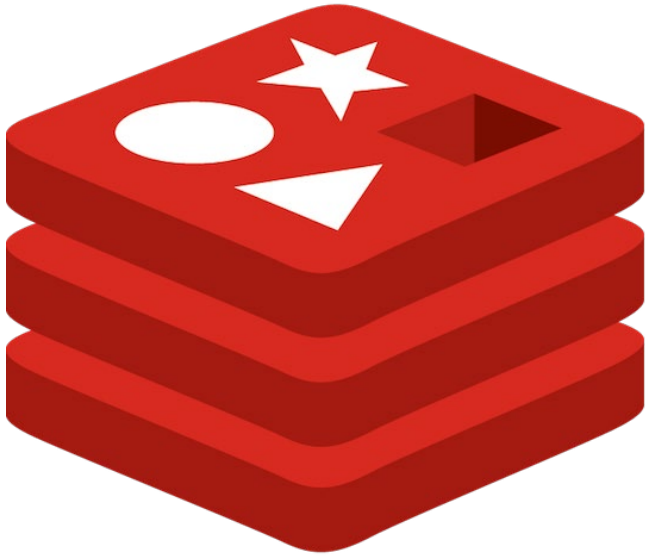
Ranked by DB-Engines
https://db-engines.com/en/ranking

Redis? – Hash-Structured



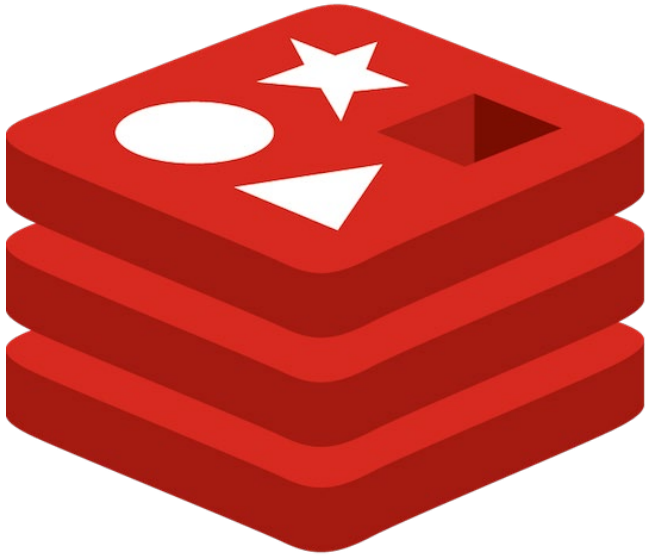
- 내부 Database는 Hash-Structured로 구성됨
  - hashFunction(Key)에 해당하는 index에 Key-Value Pair(Entry)를 저장함
  - Auto resizable
  - Incremental rehashing
    - Hash를 Resize할 때, 요청 처리 속도에 지장이 가지 않게 점차적으로 Rehashing을 진행함

Strongness & Weakness



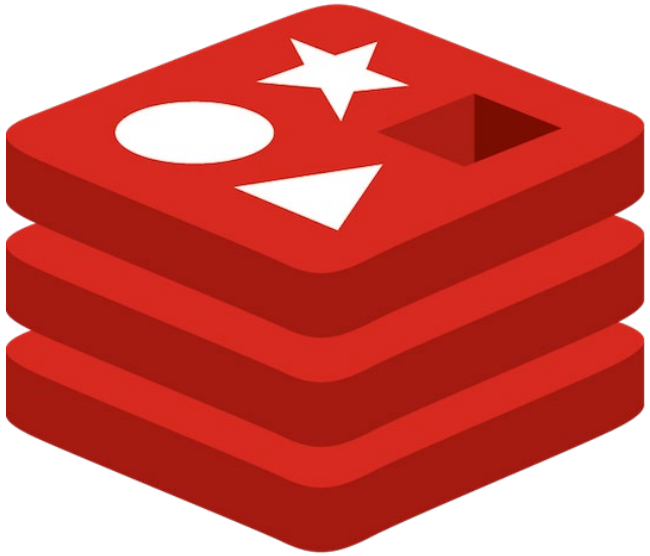
- Strongness
  - In-Memory Database이므로 처리속도가 굉장히 빠름
    - Memory: 0.1μs (평균처리속도)
    - Disk: 10ms
  - 메신저 서비스, 실시간 분석에 사용하기 좋음
  - 대부분의 프로그래밍 언어에서 사용 가능
    - C, C++, Java, Python, Javascript, …
  - Key-Value Database이므로 간단한 형태의 데이터를 저장하기 좋음
    - Machine Learning 데이터
    - IoT 데이터
    - …

Strongness & Weakness



- Weakness
  - 메모리 사이즈보다 큰 데이터 SET을 저장할 수 없음
  - In-Memory Database이므로 데이터 지속성을 유지하는데 비용이 많이 소모
    - 디스크에 로그를 작성하는 방식으로 해결함
    - RDB / AOF log methods
  - 로그 작성 시, 요청 처리 속도에 영향을 줄 수 있기 때문에 복잡한 데이터 혹은 대용량 데이터를 운용하기에는 적합하지 않음
    - 관계형 데이터

Features

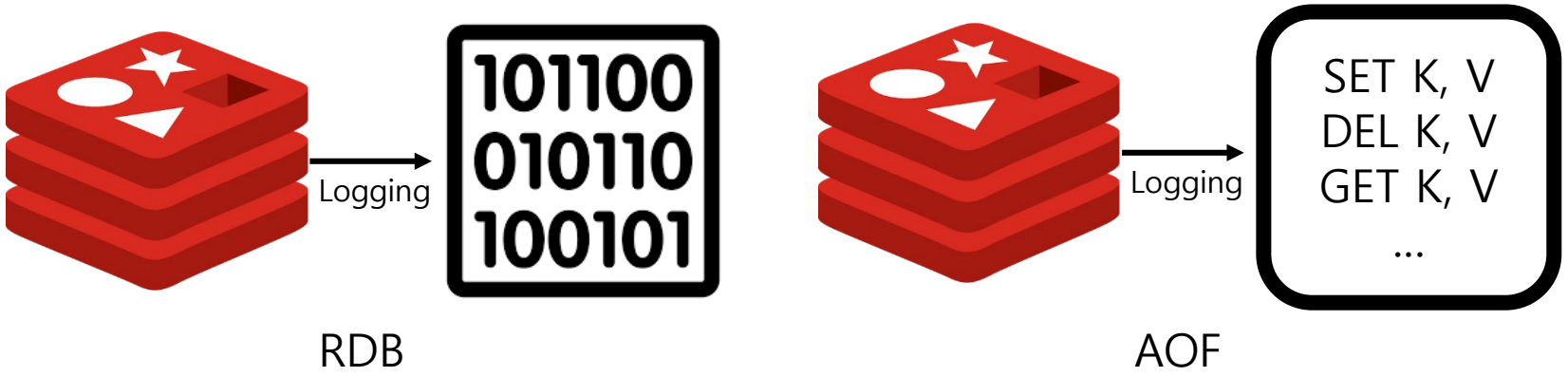


- Persistent Methods
  - RDB
  - Append-only Log (AOF)
- Clustering
  - Data Sharding
  - Master-Slave Replication
- etc…

Features - Persistent Methods

- Redis는 In-Memory Database이므로, 서버가 비정상적으로 종료되면 데이터가 날아갈 위험이 있음
  - 별도의 Log를 디스크에 저장함으로써 데이터 지속성을 유지함

- RDB
  - Redis에 저장된 데이터들을 Binary Dump 형식 Log로 저장하는 방식
- Append-only Log (AOF)
  - Redis에서 수행된 Command들을 Log로 저장하는 방식

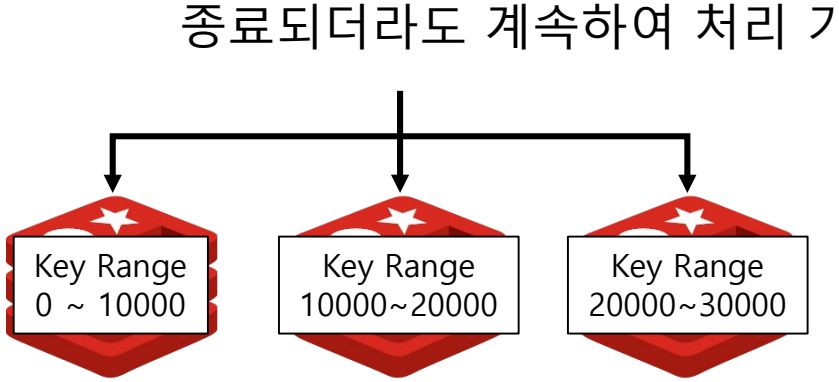


RDB AOF

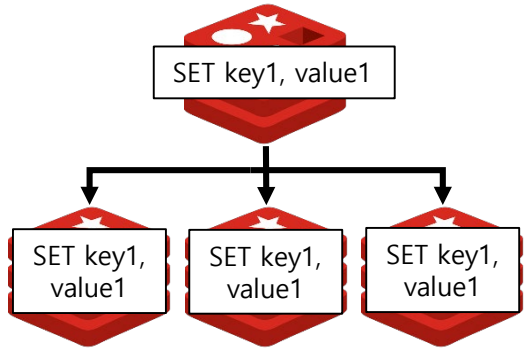
Features – Clustering

- Redis는 분산 시스템을 구성하기 위해 Clustering을 제공하고 있음

- Clustering으로 구성하여 다음과 같은 이점을 얻을 수 있음
  - Load Balancing (Data Sharding)
    여러 Redis node에 Data를 분산하여 저장
  - Fault Tolerance (Master-slave Replication)
    데이터 처리 중에 Redis node에 failure가 발생하여 비정상적으로 종료되더라도 계속하여 처리 가능

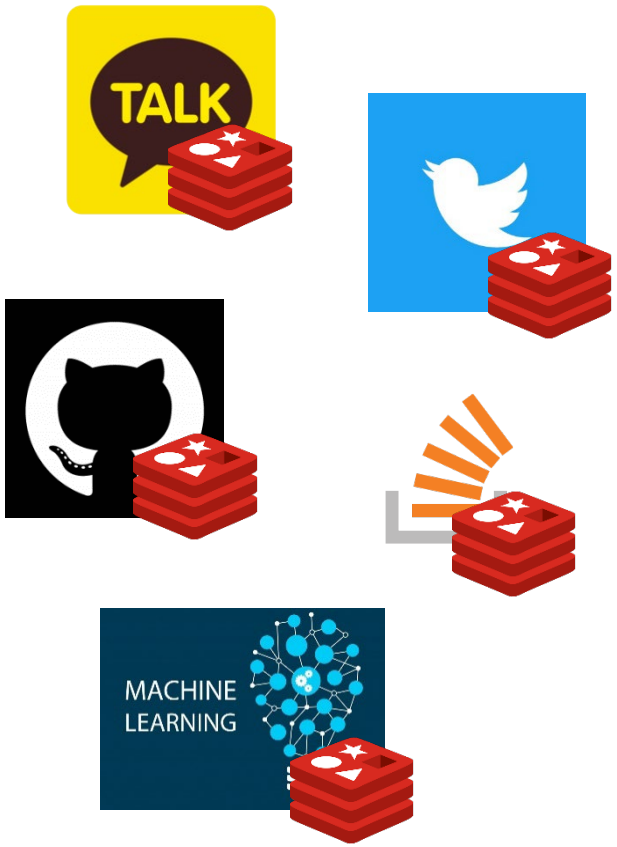


Load Balancing
(Data Sharding)



Fault Tolerance
(Master-slave Replication)

Applications



- 많은 회사에서 상용으로 사용하고 있음
  - Kakao
  - Twitter
  - Github
  - StackOverflow
  - Airbnb
  - …
- In-Memory Database이므로 빠른 처리 성능이 필요한 상황에서 많이 사용
  - 메신저
    - KakaoTalk
    - Twitter
  - Real-time Data Analysis
    - Github, StackOverflow, Airbnb
- Machine Learning에서 데이터를 저장할 때에도 사용

RocksDB

A Persistent Key-Value Store for Fast Storage Environments

연세대학교
YONSEI UNIVERSITY

RocksDB Introduction

- RocksDB?
  - Facebook에서 오픈소스로 공개 (Licensed under the Apache License, Version 2.0.)
  - 2012년 5월, Google에서 개발한 LevelDB(v1.5)를 기반으로 한 Key-Value 저장소
  - C++로 개발, 최근 Java 라이브러리도 제공
  - 멀티 코어 환경의 SSD 저장 장치 기반 서버에 최적화
  - Pluggable 데이터베이스
  - Storage Engine으로 각광 받음
    - Ex) MyRocks, MongoRocks …

- 특징
  - LSM-Tree 구조 사용
  - Column Family
  - BloomFilter를 활용하여 Read성능 개선
  - Transaction 지원
  - Time to Live 데이터 지원

RocksDB History

- 2012
  Project Start(May)
  Initial release – 1.4.0.fb(Jun 19)
- 2013
  RocksDB Wiki Open(Jan 4)
- 2014
  RocksDB Blog Open(March 27)
  RocksDB 3.0 Release(May 19)
- 2016
  RocksDB 4.2 Release(Feb 24)
- 2018
  RocksDB 5.10.2 Release(Feb 5)

*RocksDB관련 사이트 : https://rocksdb.org

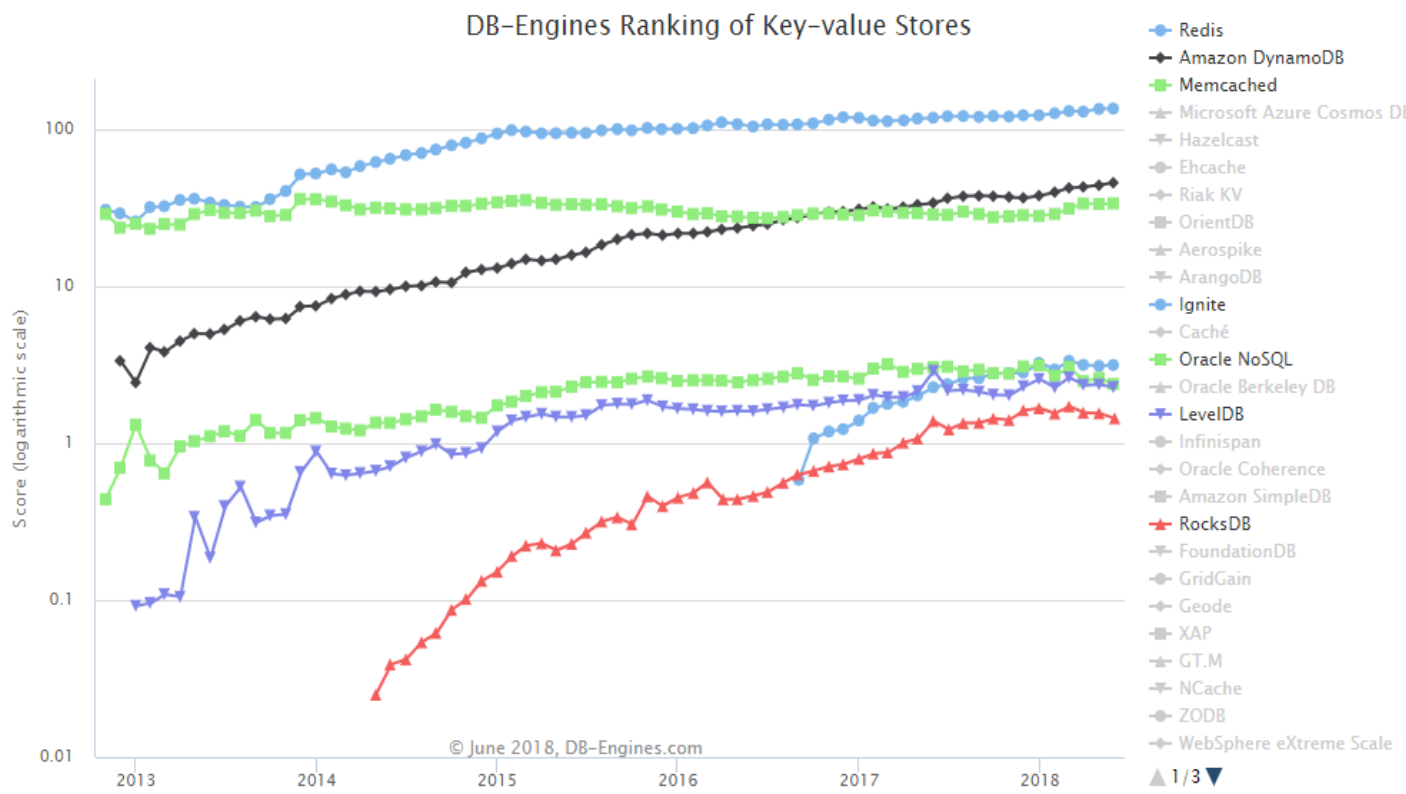
Key-Value Store Market Share

67 systems in ranking, June 2018

Rank Jun 2018	Rank May 2018	Rank Jun 2017	DBMS	Database Model	Score Jun 2018	Score May 2018	Score Jun 2017
1.	1.	1.	Redis	Key-value store	136.30	+0.95	+17.42
2.	2.	2.	Amazon DynamoDB	Multi-model	45.79	+1.60	+11.78
3.	3.	3.	Memcached	Key-value store	33.80	+0.25	+5.07
4.	4.	↑ 7.	Microsoft Azure Cosmos DB	Multi-model	19.20	+1.66	+12.82
5.	5.	↓ 4.	Hazelcast	Key-value store	9.09	-0.12	+0.48
6.	6.	↓ 5.	Ehcache	Key-value store	6.77	-0.17	-0.65
7.	7.	↓ 6.	Riak KV	Key-value store	6.26	+0.01	-1.06
8.	8.	8.	OrientDB	Multi-model	5.35	+0.10	-0.46
9.	9.	9.	Aerospike	Key-value store	4.07	+0.01	-0.21
10.	10.	10.	ArangoDB	Multi-model	3.52	-0.18	+0.44
11.	11.	↑ 18.	Ignite	Multi-model	3.15	+0.04	+0.88
12.	↑ 13.	↑ 13.	Caché	Multi-model	2.42	+0.01	-0.56
13.	↓ 12.	↓ 11.	Oracle NoSQL	Key-value store	2.39	-0.22	-0.67
14.	14.	↓ 12.	Oracle Berkeley DB	Multi-model	2.31	-0.09	-0.67
15.	15.	15.	LevelDB	Key-value store	2.27	-0.09	-0.62
16.	↑ 17.	16.	Infinispan	Key-value store	2.19	-0.03	-0.42
17.	↓ 16.	↓ 14.	Oracle Coherence	Key-value store	2.02	-0.23	-0.89
18.	18.	↓ 17.	Amazon SimpleDB	Key-value store	1.81	-0.22	-0.58
19.	19.	19.	RocksDB	Key-value store	1.45	-0.11	+0.06
20.	↑ 21.		FoundationDB	Multi-model	1.08	-0.04	

Ranked by Key-Value Store
https://db-engines.com/en/ranking/key-value+store

Key-Value Store Market Share



Ranked by Key-Value Store
https://db-engines.com/en/ranking/key-value+store

RocksDB

- *Disk-based key-value database*
- *Use Log-structured Merge-tree (**LSM-tree**)*

LSM-Tree

- ***Write amplification (WA)***
  - *Decrease data processing performance*
  - *Decline the lifespan*
- ***Space amplification (SA)***
  - *Increasing space usage*

Reduce WA and SA by tuning RocksDB knobs

- *Too many factors for performance tuning*
  - *Knobs, workload, hardware*

RocksDB LSM-Tree & WA, SA

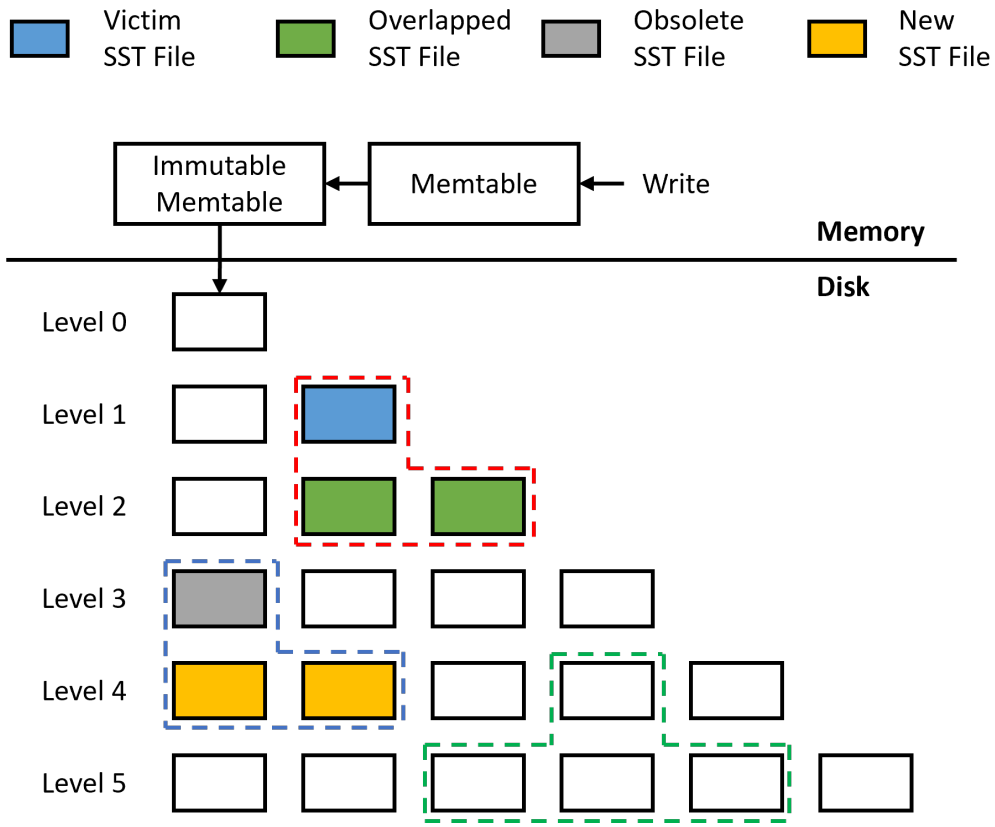
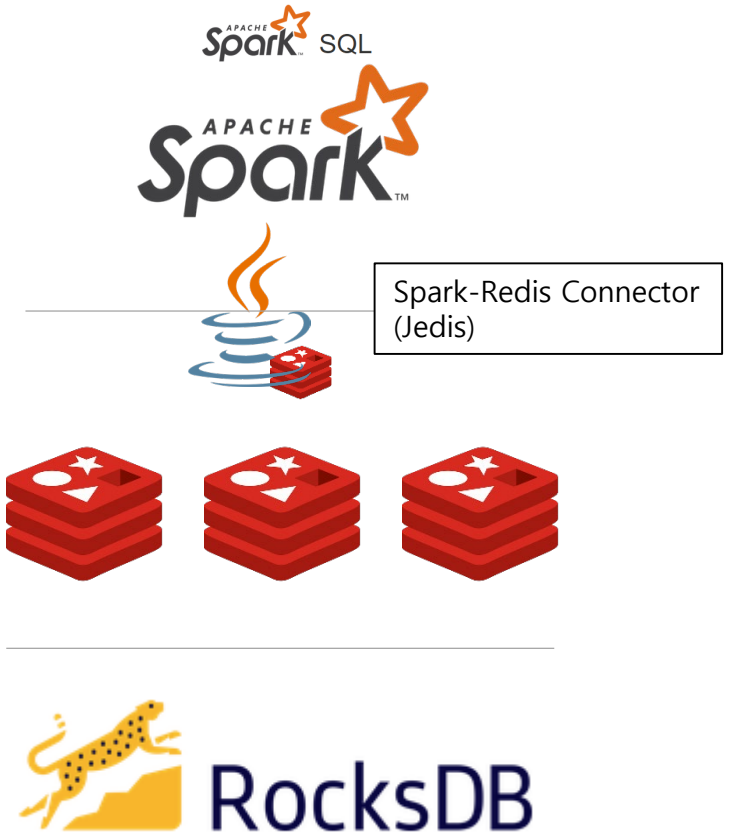


Figure 1. LSM-Tree and compaction

- **Write Amplification (WA)**
  - Additional write operations
  - **Overlapped SST File**
- **Space Amplification (SA)**
  - Additional space occupancy
  - **Obsolete SST File**
- **Issues**
  - Multi-threaded
  - Important to reduce **WA** and **SA**

StarLab Overall Structure

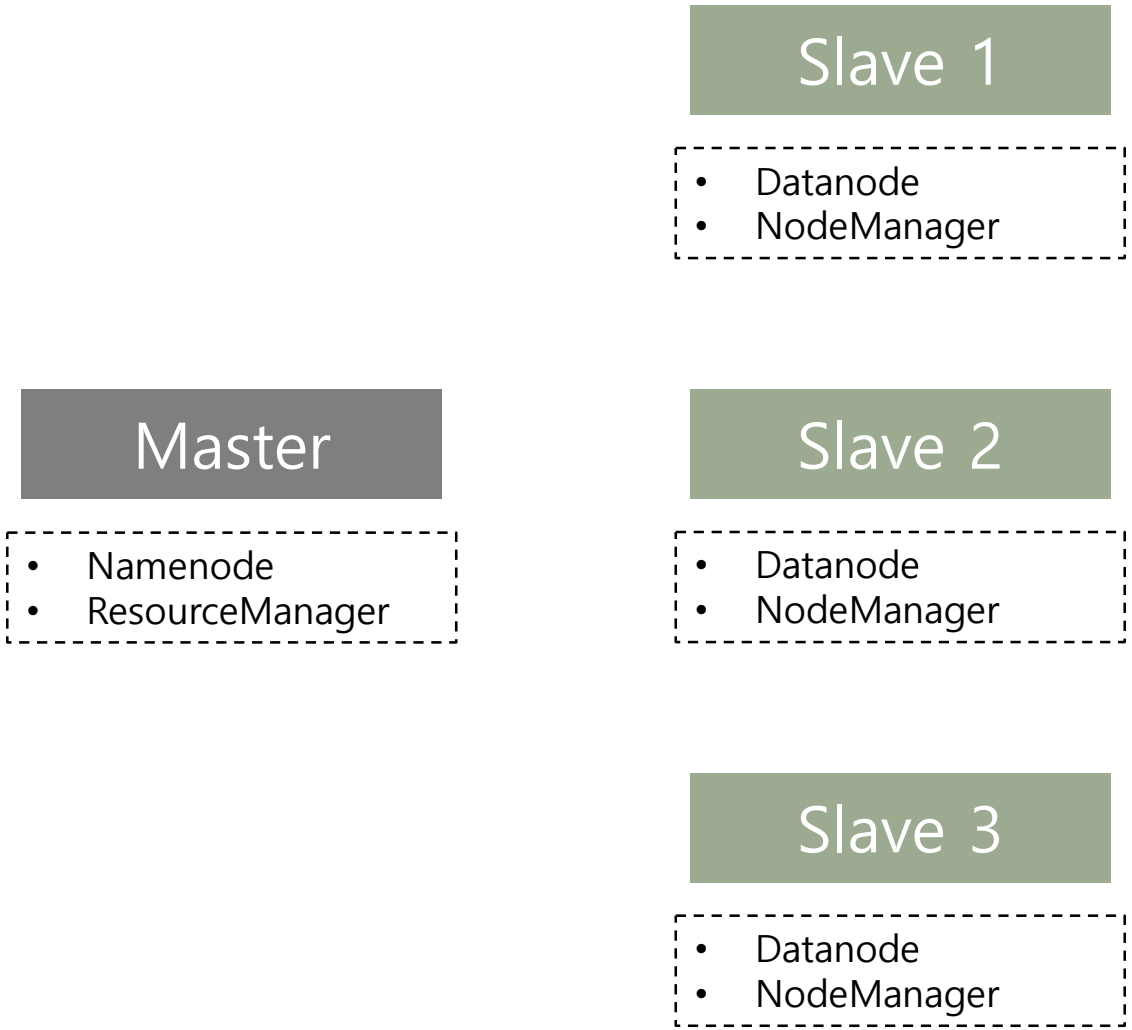


ADDB

상태	이름 ↑	영역	권장사항	다음에서 사용 중:	내부 IP	외부 IP	연결
✓	master	asia-northeast3-a			10.178.0.3 (nic0)	34.64.225.233 (nic0)	SSH
✓	slave-1	asia-northeast3-a			10.178.0.4 (nic0)	34.64.102.215 (nic0)	SSH
✓	slave-2	asia-northeast3-a			10.178.0.5 (nic0)	34.64.110.245 (nic0)	SSH
✓	slave-3	asia-northeast3-a			10.178.0.6 (nic0)	34.64.97.252 (nic0)	SSH
○	test-slave	asia-northeast3-a			10.178.0.14 (nic0)		SSH

SW스타랩 ADDB

- Hadoop - 빅데이터의 분산 저장과 분석을 위한 분산 컴퓨팅 솔루션
  - HDFS + Yarn



SW스타랩 ADDB

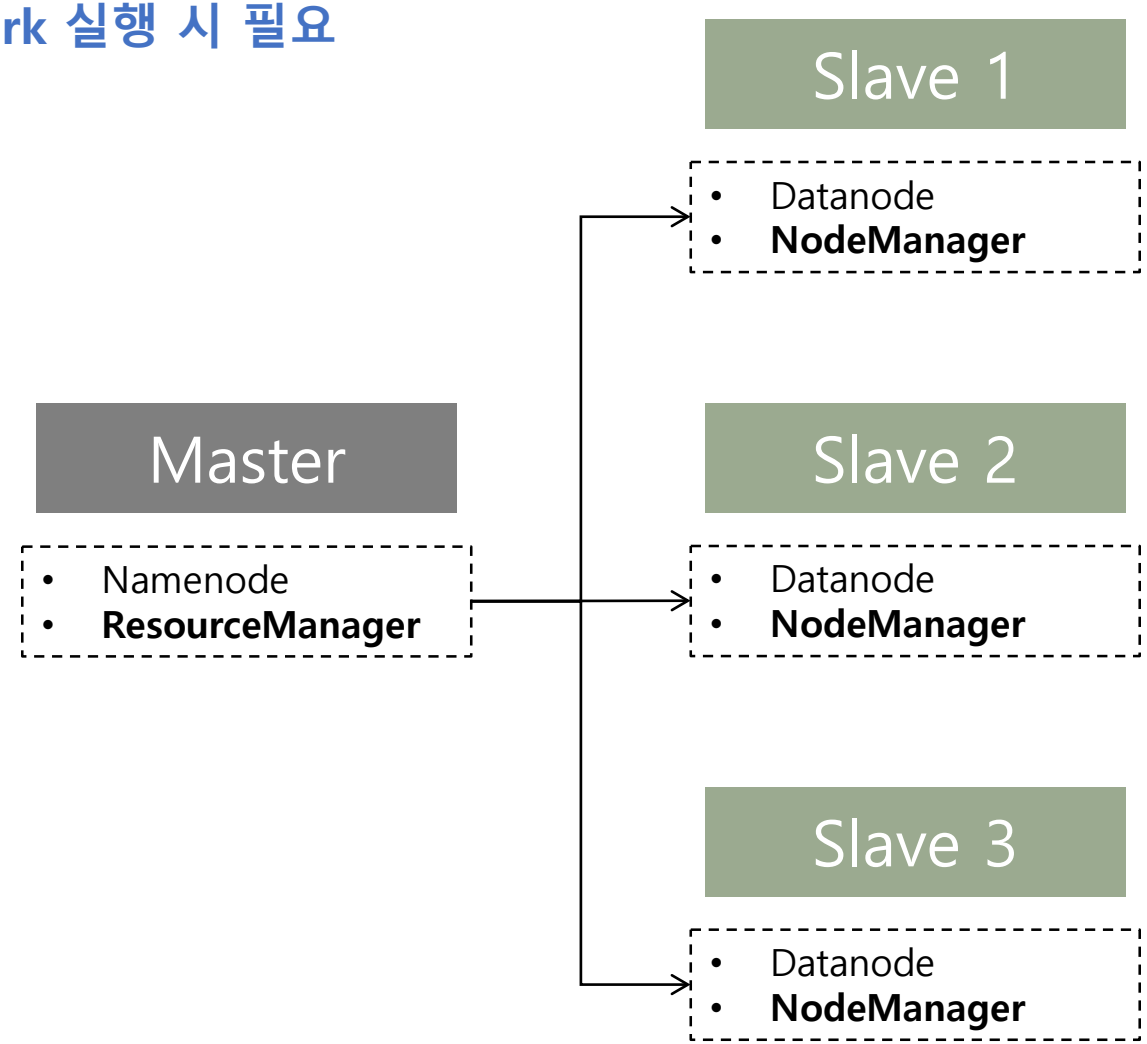
- Hadoop - 빅데이터의 분산 저장과 분석을 위한 분산 컴퓨팅 솔루션
  - **HDFS(Hadoop Distributed File System)** + Yarn

- Data를 분산하여 저장

SW스타랩 ADDB

- Hadoop - 빅데이터의 분산 저장과 분석을 위한 분산 컴퓨팅 솔루션
  - HDFS + **Yarn(Yet Another Resource Negotiator)**

- Spark 실행 시 필요



SW스타랩 ADDB

- Hadoop - 빅데이터의 분산 저장과 분석을 위한 분산 컴퓨팅 솔루션
  - **HDFS(Hadoop Distributed File System)** + Yarn

- HDFS 대신 ADDB 사용

SW스타랩 ADDB

- Hadoop - 빅데이터의 분산 저장과 분석을 위한 분산 컴퓨팅 솔루션
  - **ADDB**+ Yarn