20220425

적제

수집한 데이터를 어디에, 어떻게 저장할 것인가를 다룬다.

수집한 데이터는 특징에 따라 처리 방식과 적재 위치가 달라질 수 있다.

적제는 빅데이터 시스템의 중심에 위치해 중요한 만큼, 관련 소프트웨어가 다양하면서 기술 복잡도도 매우 높은 편이다.

실시간 로그는 4K정도로 4K보다 커지면 곤란해질 수 있다.

하둡

1. 대용량 데이터를 분산 저장
2. 분산 저장된 데이터를 가공/분석 처리하는 기능

하둡의 맵리듀스

분산컴퓨터 기술을 이해하는 중요한 열쇠로서, 컴퓨터에 분산 저장되어 있는 데이터로부터 어떻게 효율적으로 일을 나눠서 실행시킬 수 있느냐고, 다음으로 여러 컴퓨터가 나눠서 실행한 결과들을 어떻게 하나로 모으냐라는 것을 편리하게 지원하는 프레임워크이다.

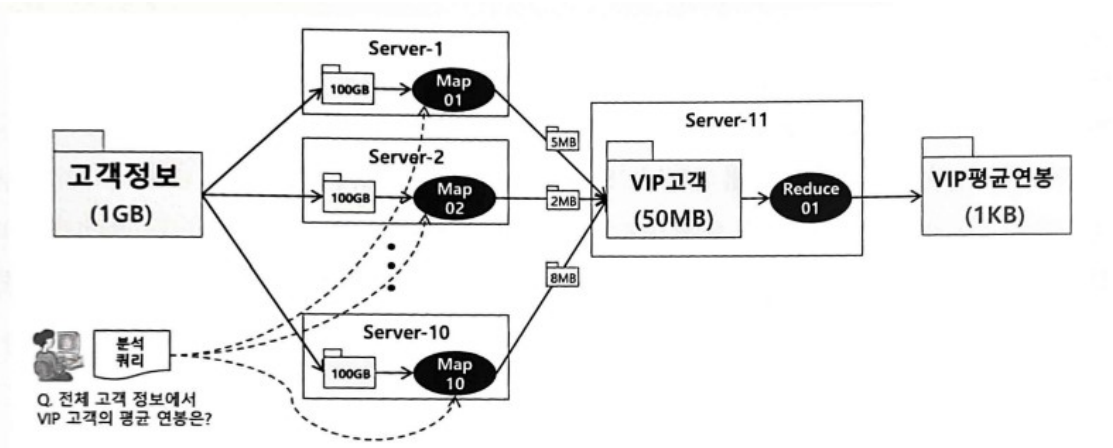


그림 4.4 MapReduce 기본 구조

❶ 고객정보가 담긴 1GB의 파일을 100MB 파일 10개로 나눠서 10대의 서버(하둡 데이터노드)에 분산 저장(나눠진 100MB 파일을 블록 파일이라 부르며, 일반적으로 128MB 블록 단위로 처리)

❷ 전체 고객정보에서 VIP 고객의 평균연봉 조회 쿼리를 실행, 10대의 서버에 분산 저장된 100MB의 고객정보 파일로부터 Map 프로그램이 각각 생성

❸ 실행된 Map 프로그램은 100MB의 고객정보 파일에서 VIP고객 정보만 추출한 후, 작아진 파일(2~8MB) 크기로 Server-11(Reduce)로 전송

❹ Server-11에서 Reduce 프로그램이 실행되어 Server-01(Map01)~Server-10(Map02)이 전송한 VIP 고객정보를 머지(50MB)해 평균을 구하고 결과 파일(1KB)을 생성

1 ~ 4과정은 대용량 데이터에 대한 처리를 여러 대의 서버들이 나누어 작업함으로써 한 대의 고성능 서버가 처리하기 힘든 작업을 신속하게 처리한다. 실제 MapReduce가 동작할 때는 한 대의 서버에 여러 개의 블록 파일이저장되기도 하며, 여러 개의 Map/Reduce가 한 서버에서 동시 다발적으로 실행되기도 한다. 각 Map/Reduce가실행되는 중에 중간 파일을 만들어 로컬 디스크에 임시 저장해 처리가 완료된 파일은 다음 작업 서버로 전송한다.
MapReduce 프로그램에서는 내부적으론 Split, Spil, Sort, Parition, Fetch, Shufile, Merge 등 다양한 메커니즘들이 작동하며 이 과정을 잘 이해하고 있어야 분산 환경에서 발생하는 다양한 문제에 빠르게 대처할 수 있다.

하둡의 기본 구성요소

주요 구성 요소	DataNode		블록(64MB or 128MB 등) 단위로 분할된 대용량 파일들이 DataNode의 디스크에 저장 및 관리
	NameNode		DataNode에 저장된 파일들의 메타 정보를 메모리상에서 로드해서 관리
	EditsLog		파일들의 변경 이력(수정, 삭제 등) 정보가 저장되는 로그 파일
	FsImage		NameNode의 메모리상에 올라와 있는 메타 정보를 스냅샵 이미지로 만들어 생성한 파일
	Ver. 1.x	SecondaryNameNode	NameNode의 FsImage와 EditsLog 파일을 주기적으로 유지 관리해 주는 체크포인팅 노드
		MapReduce v1	DataNode에 분산 저장된 파일이 스플릿(Map)되어 다양한 연산(정렬, 그루핑, 집계 등)을 수행한 뒤 그 결과를 다시 병합(Reduce)하는 분산 프로그래밍 기법
		JobTracker	맵리듀스의 잡을 실행하면서 태스크에 할당하고, 전체 잡에 대해 리소스 분배 및 스케줄링
		TaskTracker	JobTracker가 요청한 맵리듀스 프로그램이 실행되는 태스크이며, 이때 맵 태스크와 리듀스 태스크가 생성
주요 구성 요소	Ver. 2.x	Active/Stand-By NameNode	NameNode를 이중화해서 서비스 중인 Active NameNode와 실패 처리를 대비한 Standby NameNode로 구성
		MapReduce v2 / YARN	하둡 클러스터 내의 자원을 중앙 관리하고, 그 위에 다양한 애플리케이션을 실행 및 관리가 가능하도록 확장성과 호환성을 높인 하둡 2.x의 플랫폼
		ResourceManager	하둡 클러스터 내의 자원을 중앙 관리하면서, 작업 요청 시 스케줄링 정책에 따라 자원을 분배해서 실행시키고 모니터링
		NodeManager	하둡 클러스터의 DataNode마다 실행되면서 Container를 실행시키고 라이프 사이클을 관리
		Container	DataNode의 사용 가능한 리소스(CPU, 메모리, 디스크 등)를 Container 단위로 할당해서 구성
		ApplicationMaster	애플리케이션 실행되면 ApplicationMaster가 생성되며 ApplicationMaster는 NodeManager에게 애플리케이션이 실행될 Container를 요청하고, 그 위에서 애플리케이션을 실행 및 관리
		JournalNode	3개 이상의 노드로 구성되어 EditsLog를 각 노드에 복제 관리하며 Active NameNode는 EditsLog에 쓰기만을 수행하고 Standby NameNode는 읽기만을 실행

하둡의 활용방안

일 단위로 분리 적재된 데이터는 일/주/월/년별로 분석을 효율적으로 수행할 수 있고,

데이터를 적재해야하는 경우 전체 데이터가 아닌 해당 파티션의 데이터만 재적제할 수 있다는 장점이 있다. 파일럿 환경에서는 이러한 일련의 작업을 처리하기 위해서 주로 하이브를 사용한다.

주키퍼

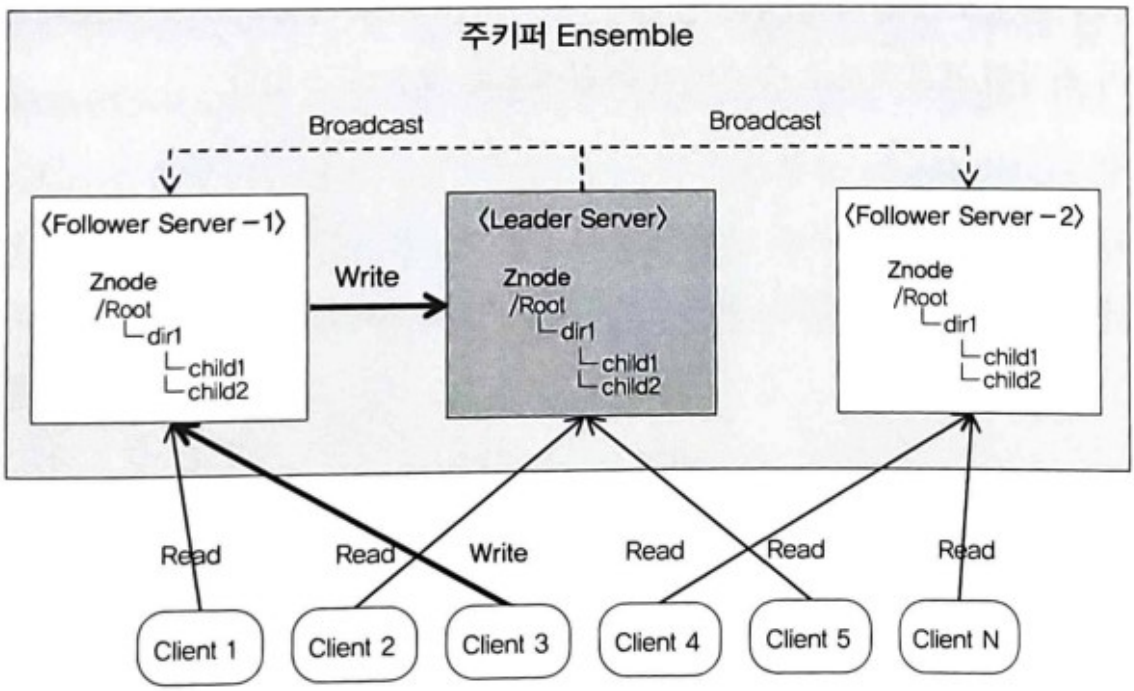
수십 ~ 수천 대의 서버에 설치돼 있는 빅데이터 분산 환경을 더욱 효율적으로 관리하기 위해서는 서버간의 정보를 쉽고 안전하게 공유해야한다. 공유된 정보를 이용해 서버간의 중요한 이벤트를 관리하면서 상호작용을 조율해 주는 코디네이터 시스템이 필요한데, 이것이 바로 분산 코디네이터인 아파치 주키퍼다.

주키퍼의 기본 요소

주요 구성 요소	Client	주키퍼의 ZNode에 담긴 데이터에 대한 쓰기, 읽기, 삭제 등의 작업을 요청하는 클라이언트
주요 구성 요소	ZNode	주키퍼 서버에 생성되는 파일시스템의 디렉터리 개념으로, 클라이언트의 요청 정보를 계층적으로 관리(버전, 접근 권한, 상태, 모니터링 객체 관리 등의 기능 지원)
	Ensemble	3대 이상의 주키퍼 서버를 하나의 클러스터로 구성한 HA 아키텍처
	Leader Server	Ensemble 안에는 유일한 리더 서버가 선출되어 존재하며, 클라이언트의 요청을 받은 서버는 해당 요청을 리더 서버에 전달하고, 리더 서버는 모든 팔로워 서버에게 클라이언트 요청이 전달되도록 보장
	Follower Server	Ensemble 안에서 한 대의 리더 서버를 제외한 나머지 서버로서, 리더 서버와 메시지를 주고받으면서 ZNode의 데이터를 동기화하고 리더 서버에 문제가 발생할 경우 내부적으로 새로운 리더를 선출하는 역할을 수행
라이선스	Apache	
유사 프로젝트	Chubby, Doozerd, Consul	

주키퍼 아키텍처

주키퍼는 3대 이상의 홀수 개의 서버로 구성되어야 하며, 그 중 반드시 1대는 리더 서버가 되고 나머지 서버는 팔로워 서버가 된다.



팔로워 서버 1에 저장된 ZNode 정보는 리더 서버에 전달되고, 리더 서버는 다른 모든 팔로워 서버에 요청받은 ZNode 정보를 브로드캐스트한다.

주키퍼 활용 방안

파일럿 프로젝트에서 사용하는 하둡, HBase, 카프카, 스톰의 내부에서 주키퍼에 의존해 클러스터 멤버십 기능 및 환경설정의 동기화 등을 사용하고 있어 없어서는 안 될 중요 소프트웨어다.

적재 파일럿 실행 단계

1단계 - 적재 아키텍처

요구사항 구체화 및 분석

적재 요구사항 구체화	분석 및 해결 방안
1. 100대에 달하는 스마트카들의 상태 정보가 일 단위로 취합되어 제공된다.	플럼에서 수집 발생 시점의 날짜를 HdfsSink에 전달해서 해당 날짜 단위로 적재
2. 매일 100대의 스마트카 상태 정보는 약 100MB 정도이며, 220만 건의 상태 정보가 발생한다.	1년 적재 시 8억 건 이상의 데이터가 적재되며, 연 단위 분석에 하둡의 분산 병렬 처리 사용
3. 스마트카의 상태 정보 데이터의 발생일과 수집/적재되는 날짜가 다를 수 있다.	수집/적재되는 모든 데이터마다 데이터 발생일 외에 수집/적재 처리돼야 하는 처리일을 추가
4. 적재된 스마트카들의 상태 정보를 일 · 월 · 년 단위로 분석할 수 있어야 한다.	HDFS에 수집 일자별로 디렉터리 경로를 만들어서 적재
5. 적재 및 생성되는 파일은 HDFS의 특성을 잘 고려해야 한다.	플럼의 HdfsSink의 옵션을 파일럿 프로젝트의 HDFS에 최적화해서 설정
6. 적재가 완료된 후에는 원천 파일이 삭제돼야 한다.	플럼의 Source 컴포넌트 중 SpoolDir의 DeletePolicy 옵션을 활용

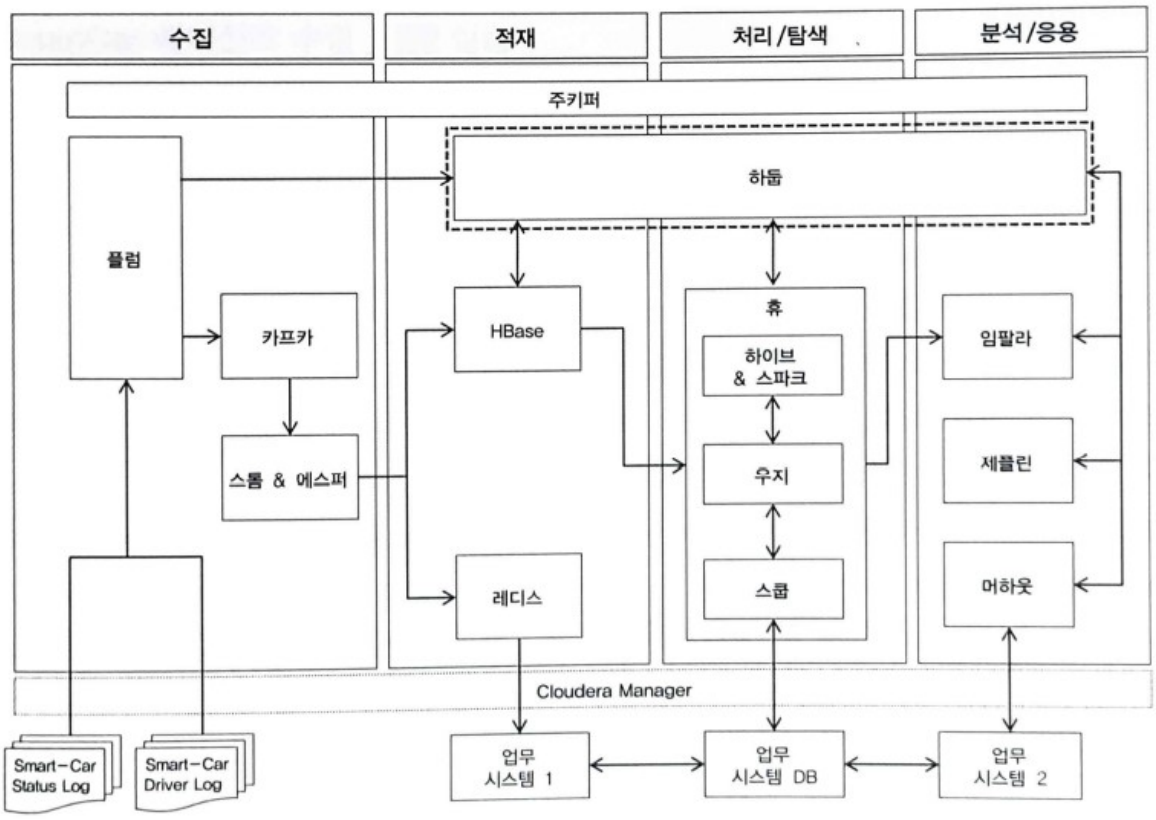
적재 아키텍처

HDFS에 적재된 데이터는 부분 수정/삭제가 어렵기 때문에 유형에 따라 특별한 관리 정책이 필요하다. 시계열 형식의 트랜잭션(거래, 이력 등) 데이터는 일자별 파티션 폴더를 구성해 파티션 단위로 데이터를 적재 및 수정하며, 마스터(고객정보, 상품정보 등) 데이터는 상대적으로 크기가 작아 전체 데이터셋을 교체해 버리는 방식을 주로 이용한다.

HDFS의 파티션 적재

HDFS의 적재 경로를 하이브에서 인지할 수 있는 특정한 구분값(날짜, 시간, 코드 등)으로 파티셔닝한다.

2단계 - 적재 환경 구성



하둡의 상태 정보 확인 : server01.hadoop.com:9870

잡 모니터링 관련 기능에 문제가 발생할 경우 CM 홈의 [YARN (MR2Include)] → [인스턴스]에서 JobHistory Server의 상태가 [시작됨] 상태인지 확인하고, 정지 상태이면 재시작한다.

리소스 매니저: http://server01.hadoop.com:8088/cluster

(Job) 히스토리: http://server01.hadoop.com:19888/jobhistory

로그 시뮬레이터 작동

2022년 4월 25일의 100대의 스마트카에 상태 정보 로그 생성

java -cp bigdata.smartcar.loggen-1.0.jar
com.wikibook.bigdata.smartcar.loggen.CarLogMain
20220425 100 &

내용 확인

/home/pilot-pjt/working/SmartCar 경로에 가서

tail -f SmartCarStatusInfo_20220425.txt를 이용하여 확인

kill -9 Pid(Process ID) 로그 삭제

빅데이터 실시간 적재 개요

실시간 로그 분석에서는 작지만 대량으로 발생하는 메시지성 데이터를 실시간으로 분석(집계, 분류, 관계 등) 처리하며, 해당 결과를 인메모리에 저장해 주변 시스템과 빠르게 공유한다.

이때 대량의 메시지 데이터를 영구 저장하기 위해서 하둡을 직접 이용하지는 않는다. 이유는 초당 수천건의 트랜잭션이 발생하는 메시지의 경우 파일 개수가 기하급수적으로 늘어나고 이로 인해 하둡 클러스터에 지나친 오버헤드가 발생하기 때문이다.

이러한 문제를 해결하기 위해 중간에 메시지를 특정 크기로 모았다가 한꺼번에 적재하거나 대규모로 트랜잭션 데이터를 처리하는 데 최적화된 칼럼 지향형 NoSQL 데이터베이스를 주로 사용한다.

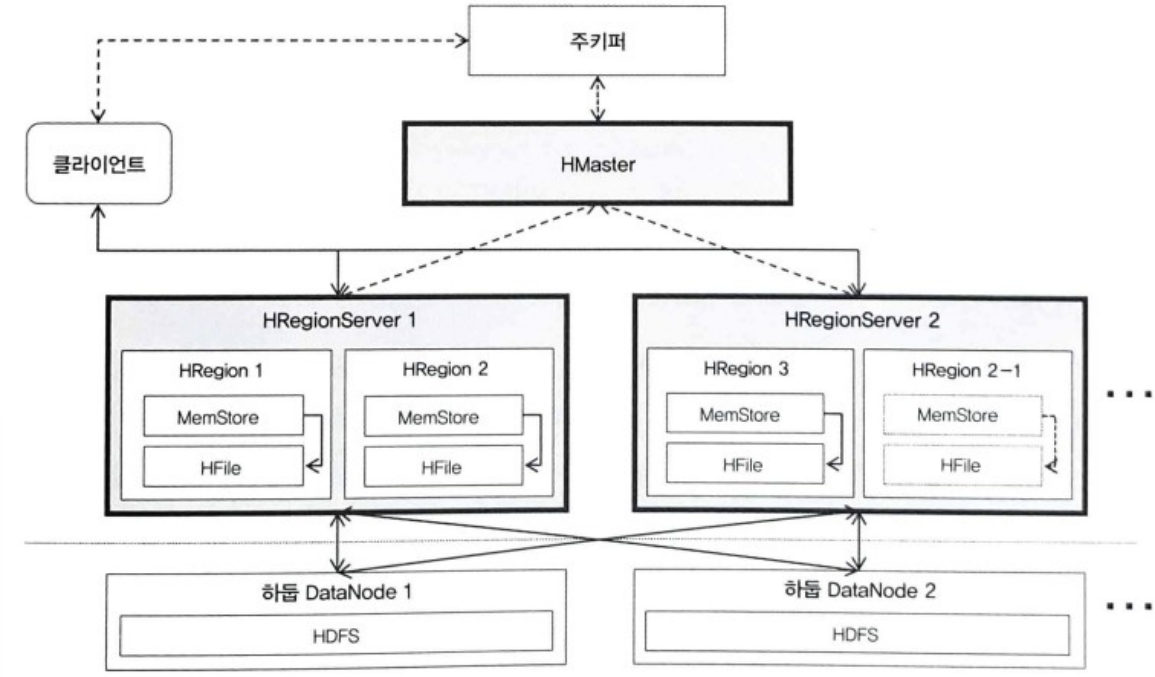
HBase

NoSQL 데이터베이스들은 데이터를 키/값 구조로 단순화하고, 칼럼 또는 도큐먼트 형식의 제양사항이 적은 스키마 모델로 만들어 고성능 쓰기/읽기가 가능하다는 공통점을 가지고 있다.

주요 구성 요소	HTable	칼럼 기반 데이터 구조를 정의한 테이블로서, 공통점이 있는 칼럼들의 그룹을 묶은 칼럼 패밀리와 테이블의 로우를 식별해서 접근하기 위한 로우키로 구성
	HMaster	HRegion 서버를 관리하며, HRegion들이 속한 HRegion 서버의 메타 정보를 관리
	HRegion	HTable의 크기에 따라 자동으로 수평 분할이 발생하고, 이때 분할된 블록을 HRegion 단위로 지정
	HRegionServer	분산 노드별 HRegionServer가 구성되며, 하나의 HRegionServer에는 다수의 HRegion이 생성되어 HRegion을 관리
	Store	하나의 Store에는 칼럼 패밀리가 저장 및 관리되며, MemStore와 HFile로 구성됨
	MemStore	Store 내의 데이터를 인메모리에 저장 및 관리하는 데이터 캐시 영역
	HFile	Store 내의 데이터를 스토리지에 저장 및 관리하는 영구 저장 영역
라이선스	Apache	
유사 프로젝트	BigTable, Cassandra, MongoDB	

HBase 아키텍처

하둡의 HDFS를 기반으로 설치 및 구성됨.



HBase와 HDFS 사이의 모든 데이터 스트림 라인들은 항상 열려있으므로 레이턴시가 발생하지 않는다.

레디스

레디스(Redis)는 분산 캐시 시스템이면서 NoSQL 데이터베이스처럼 대규모 데이터 관리 능력도 갖춘 IMDG(In-Memory Data Grid) 소프트웨어다.

영구적으로 저장할 수 있는 스냅샷 기능을 제공하며 데이터 유실에 대비해 AOF(Append Only File) 기능으로 정합성을 보장한다.

레디스의 기본 요소

주요 구성 요소	Master	분산 노드 간의 데이터 복제와 Slave 서버의 관리를 위한 마스터 서버
	Slave	다수의 Slave 서버는 주로 읽기 요청을 처리하고, Master 서버는 쓰기 요청을 처리
	Sentinel	레디스 3.x부터 지원하는 기능으로, Master 서버에 문제가 발생할 경우 새로운 Master를 선출하는 기능
	Replication	Master 서버에 쓰인 내용을 Slave 서버로 복제해서 동기화 처리
	AOF/Snapshot	데이터를 영구적으로 저장하는 기능으로, 명령어를 기록하는 AOF와 스냅샷 이미지 파일 방식을 지원
라이선스	BSD	
유사 프로젝트	jBoss Infinispan, MemCached, Mambase	

스톰(Storm)

스톰(Storm)은 스피드 데이터를 인메모리 상에서 병렬 처리하기 위한 소프트웨어다.

모든 데이터를 인메모리 상에서 분산 병렬 처리하고, 분산 데이터를 통제하기 위한 강력한 기능(분리, 정제, 통합, 집계 등)과 아키텍처도 제공한다.

실시간 분산 처리 유형으로는 데이터 발생과 동시에 처리하는 완전 실시간 방식과 발생한 데이터를 적제한 후 빠르게 배치를 실행하는 마이크로 배치 방식이 있다.

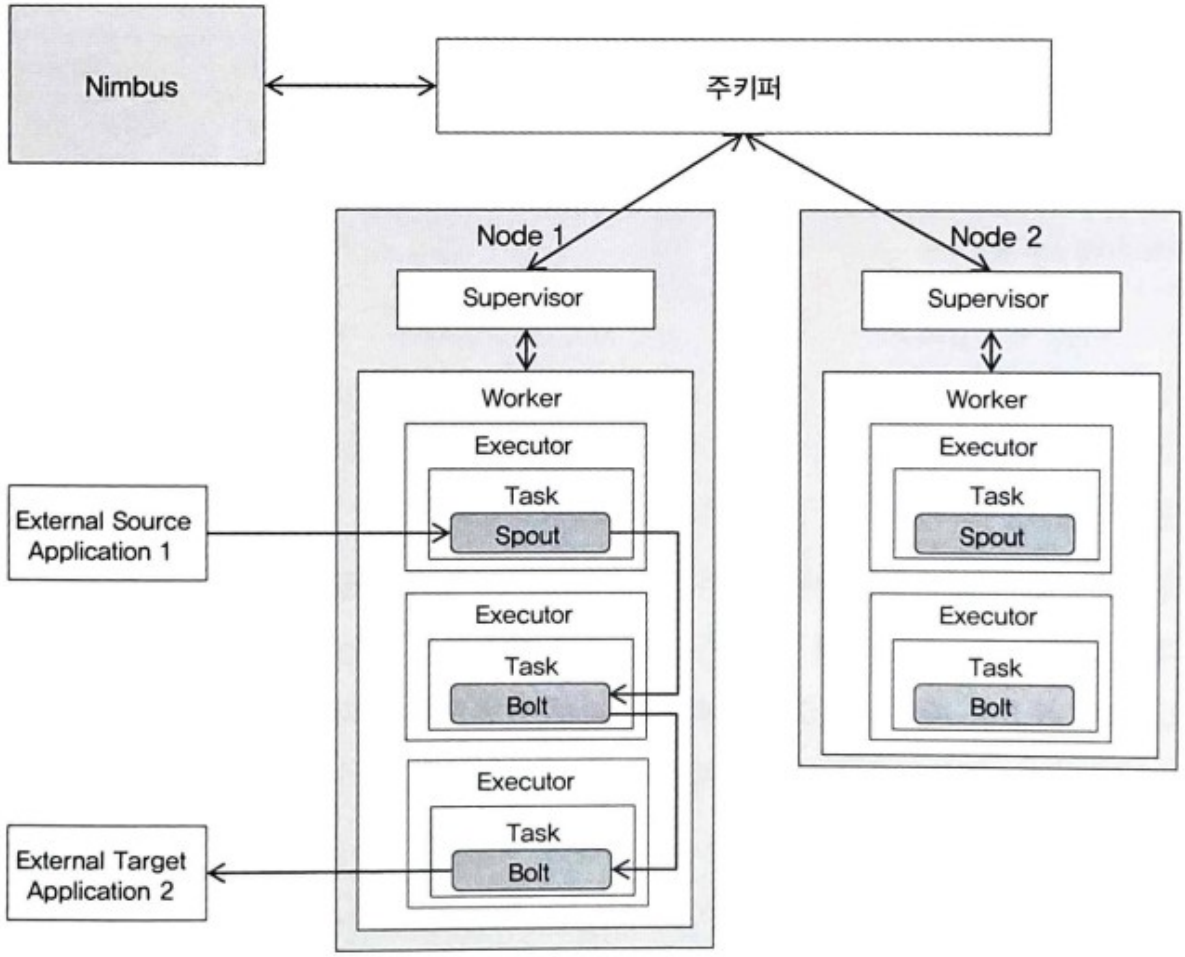
스톰의 기본 요소

주요 구성 요소	Spout	외부로부터 데이터를 유입받아 가공 처리해서 튜플을 생성, 이후 해당 튜플을 Bolt에 전송
	Bolt	튜플을 받아 실제 분산 작업을 수행하며, 필터링(Filtering), 집계(Aggregation), 조인(Join) 등의 연산을 병렬로 실행
	Topology	Spout-Bolt의 데이터 처리 흐름을 정의, 하나의 Spout와 다수의 Bolt로 구성
	Nimbus	Topology를 Supervisor에 배포하고 작업을 할당, Supervisor를 모니터링하다 필요 시 페일오버(Fail-Over) 처리
	Supervisor	Topology를 실행할 Worker를 구동시키며 Topology를 Worker에 할당 및 관리
	Worker	Supervisor 상에서 실행 중인 자바 프로세스로 Spout와 Bolt를 실행
	Executor	Worker 내에서 실행되는 자바 스레드
	Tasker	Spout 및 Bolt 객체가 할당

스톰 아키텍처

스톰의 아키텍처를 이해하기 위해서는 먼저 Nimbus와 Supervisor의 역할을 알아야 한다.

Nimbus가 자바 프로그램으로 구성된 Topology jar를 배포하기 위해 주키퍼로 부터 Supervisor 정보를 알아낸다.



스톰 아키텍처는 매우 견고한 장애 복구 기능도 제공한다.

에스퍼

에스퍼(Esper)의 경우 실시간 스트리밍 데이터의 복잡한 이벤트 처리가 필요할 때 사용하는 룰 엔진이다.

에스퍼의 기본 요소

주요 구성 요소	Event	실시간 스트림으로 발생하는 데이터들의 특정 흐름 또는 패턴을 정의
	EPL	유사 SQL을 기반으로 하는 이벤트 데이터 처리 스크립트 언어
	Input Adapter	소스로부터 전송되는 데이터를 처리하기 위한 어댑터 제공 (CSV, Socket, JDBC, Http 등)
	Output Adapter	타깃으로 전송하는 데이터를 처리하기 위한 어댑터 제공 (HDFS, CSV, Socket, Email, Http 등)
	Window	실시간 스트림 데이터로부터 특정 시간 또는 개수를 설정한 이벤트들을 메모리 상에 등록한 후 EPL을 통해 결과를 추출

간혹 오픈소시이지만 상업적 활용이 어렵고 소스 공개 의무가 있어 컴플라이언스 이슈가 발생하는 라이선스가 있다. 대표적으로 GPL, AGPL 등이 있는데 에스파가 GPL2.0 라이선스를 체택하고 있어 주의할 필요가 있다.

에스퍼 아키텍처

에스퍼는 CEP(Complex Event Processing) 엔진이다. 여기서 엔진은 단순 자바 라이브러리 프로그램으로, 설치와 사용은 매우 간단하다.

에스퍼의 EPL(Event Processing Language)을 이용해 대규모 분산 아키텍처를 구성할 때는 룰을 통합 관리한다.

실시간 적재 파일럿 실행

1단계 - 실시간 적재 아키텍처

실시간 적제 요구사항

- 요구사항 1 : 차량의 다양한 장치로부터 발생하는 로그 파일을 수집해서 기능별 상태를 점검한다.
- 요구사항 2 : 운전자의 운행 정보가 담긴 로그를 실시간으로 수집해서 주행 패턴을 분석한다.

요구사항 구체화 및 분석

실시간 적재 요구사항 구체화	분석 및 해결 방안
1. 1초 간격으로 발생하는 100명의 운행 정보(운행 정보 1건: 약 4KB)는 손실 없이 적재해야 한다.	카프카와 스톰을 이용해 수집한 데이터에 대해 분산 처리 및 무결성을 보장하며, 분산 처리가 완료된 데이터는 HBase에 적재
2. 적재한 운행 정보를 대상으로 조건 검색이 가능해야 하며, 필요 시 수정도 가능해야 한다.	HBase의 테이블에 적재된 데이터는 스캔 조건으로 검색하며, 저장(Put) 기능을 이용해 기적재한 데이터에 대해 칼럼 기반으로 수정
3. 운전자의 운행 정보 중 30초를 기준으로 평균 속도가 80Km/h를 초과한 정보는 분리 적재한다.	에스퍼의 EPL에서 사용자별로 운행 정보를 그루핑하고, 30초의 윈도우 타임(Window Time) 조건으로 평균 시속 집계 및 임계치별 이벤트를 정의
4. 과속한 차량을 분리 적재하기 위한 조건은 별도의 룰로 정의하고 쉽게 수정할 수 있어야 한다.	과속 기준을 80Km/h에서 100Km/h로 변경해야 할 경우 EPL의 평균 속도를 체크하는 조건값만 수정
5. 분리 적재한 데이터는 외부 애플리케이션이 빠르게 접근하고 조회할 수 있게 해야 한다.	실시간 이벤트로 감지된 데이터는 인메모리 기반 저장소인 레디스에 적재해서 외부 애플리케이션에서 빠르게 조회
6. 레디스에 적재한 데이터는 저장소의 공간을 효율적으로 사용하기 위해 1주일이 경과하면 영구적으로 삭제한다.	레디스 클라이언트 라이브러리인 제디스(Jedis) 클라이언트를 이용해 데이터 적재 시 만료(Expire) 시간을 설정해 자동으로 영구 삭제 처리

스톰의 실시간 데이터 처리

1. 스톰의 Spout가 카프카의 토픽으로부터 운전자의 실시간 운행 정보를 수신받아 첫 번째 볼트에 전송한다.
2. HBase의 테이블에는 “차량번호+발생일시”를 로우키로 해서 8개의 칼럼(발생일시, 차량 번호, 가속 페달, 브레이크 페달, 운전대 회전각, 방향지시등, 주행 속도, 주행 구역)의 구조로 모든 스마트카 운전자의 운행 정보가 적재된다.
3. 레디스에 적재될 때는 현재 날짜로 키로 해서 과속한 차량의 정보를 세트 데이터 구조로 적재한다. 절재 영속 시간은 5시간으로 하며, 이후에 만료 처리되어 메모리에서 자동으로 삭제된다.

2단계 - 실시간 적재 환경 구성

- 1. HBase 설치 - 설치 후 CM홈에서 [HBase] → [구성]을 선택하고 검색란에 “HBase Thrift Http 서버 설정”을 입력 후, “HBase(서비스 전체)” 항목을 체크하고 [변경 내용 저장] 버튼을 클릭
- 2. HBase의 서비스를 시작 또는 시작한다