분산 코디네이터 분야
Stack 통합 Test 결과보고서
[Zookeeper]

2013. 05.

nipa 정보통신산업진흥원 National IT Industry Promotion Agency
nipa 공개SW역량프라자 Open Source Software COMPENTENCE PLAZA

목 차

I. Stack 통합 테스트 개요

공개SW Stack 통합테스트는 여러 공개SW들의 조합으로 시스템 Stack을 구성한 후 Stack을 구성하는 공개SW의 상호운용성에 중점을 두고 기능 및 성능테스트 시나리오를 개발하여 테스트를 진행한다.

본 통합테스트를 통해 안정된 Stack 정보를 제공하여 민간 및 공공 정보시스템 도입 시 활용될 수 있도록 한다.

1. 목적

□ 공개SW Stack 통합 테스트 수행 목적

- o 공개SW로 구성된 Stack이 유기적으로 잘 동작함을 확인
- o 다양한 Stack 구성에 기반을 둔 테스트를 통해 안정된 Stack 조합 규명
- o 공개SW 시스템 도입을 위한 Stack 참조모델의 신뢰성 정보로 활용
- o 공개SW의 신뢰성과 범용성에 대한 사용자 인식 제고

Ⅱ. 분산 파일 시스템

분산 파일 시스템은 클라이언트 측에서 서버에 저장된 데이터에 접근하여, 마치 자신에게 저장되어 있는 데이터인 것처럼, 처리할 수 있는 클라이언트/서버 기반의 애플리케이션을 말한다.

사용자가 서버에 저장된 파일에 접근할 때, 서버는 그 파일의 복사본을 사용자에게 전달함으로써, 데이터가 처리되고 있는 중에는 사용자 컴퓨터상에 일시 저장되어 있다가, 처리가 끝나면 서버로 반환하도록 한다.

분산 파일 시스템은 개별 서버 상의 파일들 및 디렉토리 서비스들을 하나의 글로벌 디렉토리로 조직화함으로써, 원격지의 데이터 접근 방식이 장소에 따라 특정 지어지는 것이 아니라 어떤 클라이언트에서나 같도록 하는 것이 이상적이다. 글로벌 파일 시스템의 사용자들은 모든 파일들에 접근이 가능하며, 조직은 계층화 및 디렉토리 기반으로 구성된다.

Ⅲ. 분산 코디네이터

1. 분산환경과 로드밸런싱

분산 환경에서는 다양한 운영 상황과 예상하지 못한 장애 상황 등으로 인해 분산된 애플리케이션 서버들 사이의 자원 경합, 락 잠금/해제 등과 같은 복잡한 문제가 발생한다.

이런 문제를 해결하려면 많은 노력이 필요하고 알려지지 않았던 버그로 인해 어려움을 겪는다. 분산 코디네이터는 이러한 복잡한 문제를 쉽게 해결할 수 있는 로드밸런싱 기능을 제공한다.

※자원 경합 : 여러 서버가 한정된 공유 자원에 동시에 접근하는 문제

※분산 락 : 분산 환경에서 하나의 파일에 하나의 서버만 접근이 가능하도록 거는 잠금장치 역할

※로드밸런싱 : 로드밸런싱이란(Load Balancing)이란 하나의 인터넷 서비스가 발생하는 트래픽이 많을 때 여러 대의 서버가 분산처리하여 서버의 로드율 증가, 부하량, 속도저하등을 고려하여 적절히 분산처리하여 해결해 주는 서비스

2. 로드밸런싱 분야 주요 공개SW

로드밸런싱 분야 주요 공개SW로 많이 알려진 Apache군의 Zookeeper와 Zen load-balancer, Pound Load-balancer를 테스트 SW로 선정하고 공개SW 선정지표를 통하여 가장 점수가 높은 Apache Zookeeper로 테스트를 진행 하였다.

[표 III-1. 수집 분야 주요 공개SW]

제품명	Stack 환경	홈페이지	비고
Zookeeper	Linux	http://zookeeper.apache.org/	
Zen	Linux	http://www.zenloadbalancer.com/	
Pound	Linux	http://www.apsis.ch/pound	

[표 III-2. 수집 분야 주요 공개SW 선정지표 점수]

분야	세부분야	대상	항목[배점]				총점 [100]
			Document [25]	Support [25]	Product [30]	Community [20]	
분산 코디네이터	로드밸런싱	zookeeper	22.2	16.7	25	10	73.9
		zen	21.6	19.2	21.3	6.7	68.7
		pound	18.6	15.8	22.6	5.0	61.9

※ 공개SW 선정지표에 의해 선정된 공개SW가 품질/성능의 우수성을 뜻하는 것은 아님

※ [별첨1]공개SW Zookeeper 선정지표 테스트 결과

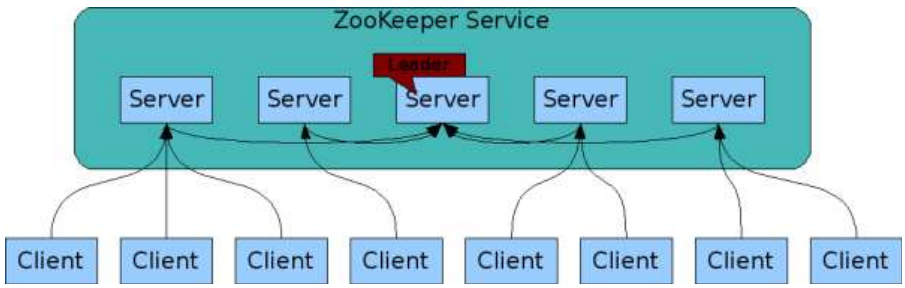
IV. 테스트 대상 소개

1. Zookeeper 소개

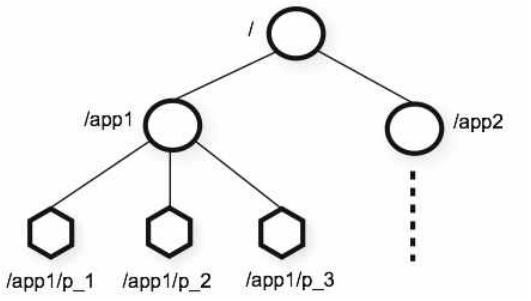
Zookeeper는 분산 코디네이터 서비스를 제공하는 아파치 오픈소스다. 분산 환경에서 락, 네이밍 서비스, 클러스터 멤버쉽 등을 쉽게 구현 할 수 있는 메커니즘을 제공한다.

최근 하둡의 서브 프로젝트에서 탈피해 메인 프로젝트로 승격을 준비하고 있다. Zookeeper(동물사육사)라하는 이름은 하둡의 서브 프로젝트가 코끼리(hadoop),거북이(Chukwa),돼지(Pig) 등과 같은 동물 이름에서 많이 따왔으며, Zookeeper의 역할은 사육사처럼 다양한 서버로 구성된 분산된 서버들을 잘 관리하고 조율하는 기능을 수행하기 때문에 지어졌다.

Zookeeper는 분산 환경을 쉽게 관리하는 기능을 제공하지만 Zookeeper 역시 여러 대의 서버로 구성된 분산 시스템이다. Zookeeper는 n개의 서버와 클라이언트 API로 구성되어 있다.



[그림 IV-1. Zookeeper 구조]



[그림 IV-2. Zookeeper 데이터 모델]

□ Client API 구성

- o create
  트리의 위치에 노드를 생성
- o delete
  노드 삭제
- o exists
  노드가 존재하는지 테스트
- o get data
  노드의 데이터를 읽어옴
- o set data
  노드의 데이터를 기록
- o get children
  노드의 자식노드 리스트를 검색
- o sync
  데이터가 전달 될 때 까지 기다림

※ 추가적인 자세한 정보는 아래의 링크 정보 참조

- http://zookeeper.apache.org/doc/r3.4.5/zookeeperOver.html#sc_designGoals

V. Stack 통합 테스트

1. 테스트 환경

□ Zookeeper 환경

[표 V-1. 테스트 SW]

SW	Version
Zookeeper	3.4.5
Java	1.6

□ Stack 환경

[표 V-2. Stack 환경]

Stack	OS	IP	SW	VERSION
A	CentOS 6.2	121.162.249.18	Zookeeper	3.4.5
B	CentOS 6.2	121.162.249.19		
C	CentOS 6.2	121.162.249.23		

□ HW 환경

[표 V-3. HW 환경]

제조사	모델명	CPU	MEM	Disk	NIC
IBM	X3550M2	Intel Xeon(R)CPU 2.40GHz	8GB	320GB	Gigabit 1Port

※ IBM 동일 사양의 HW 3대로 Hadoop Stack 구성(CentOS)

2. 주요 테스트 방법

□ 시나리오 테스트

시나리오 테스트 기법은 단일 기능에 대한 결함 여부를 확인하는 것이 아니라, 서로 다른 컴포넌트 사이의 상호작용과 간섭으로 발생할 수 있는 결함을 발견하기 위한 기법이다.

본 테스트에서는 사용자 시나리오 테스트 기법을 적용하여 Zookeeper Client API를 사용하여 Node 생성, 분산 락 구현, 세션, 로드밸런싱 등에 대한 사용자 시나리오를 도출하였다. 각각의 항목에서 도출한 세부 시나리오는 사용자가 일반적으로 수행할 수 있는 시나리오를 추출하여 테스트케이스로 작성하였다.

□ 상호 운용성 기반 테스트

상호 운용성은 서로 다른 기술로 이루어진 제품이나 서비스가 상호작용 상의 오류가 없는지 검증하는 기법으로, 본 테스트에서는 애플리케이션이 지원하는 Stack을 구성하여 애플리케이션과 Stack 환경 사이의 상호작용 상의 동작여부를 검증하였다.

3. 기능 테스트 수행 결과

기능 테스트 수행 관련 세부 시나리오는 별첨「Zookeeper 테스트 케이스」문서를 참고한다.

□ 테스트 시나리오 현황

[표 V-4. 테스트 시나리오 현황]

기능	테스트 시나리오	테스트 케이스
로드밸런싱	1	8
세션	5	5
분산 락 구현	1	8
Zookeeper Client API	8	8
시작/종료	1	3
모니터링	1	1
합 계	16	33

□ 테스트 결과

기능 테스트 시나리오를 통한 테스트 수행 결과 Zookeeper 시나리오 상의 모든 기능이 예상 결과와 동일하게 동작함을 확인하였다.

[표 V-5. 테스트 결과]

분류		PASS	FAIL	N/A
기능	개수			
로드밸런싱	8	8	0	0
세션	5	5	0	0
분산 락 구현	8	8	0	0
Zookeeper Client API	8	8	0	0
시작/종료	3	3	0	0
모니터링	1	1	0	0

4. 성능 테스트 수행

성능 테스트의 경우 하드웨어 사양뿐 아니라, OS 및 애플리케이션 환경 구성에 따라 성능 측정 결과가 상이하므로, 실제 운영 시스템 환경에 따라 테스트 결과가 다를 수 있다.

본 성능 테스트는 Zookeeper의 로드밸런싱 시스템을 Java로 구현한 뒤 구축된 멀티 서버에 등록 후 가동되는 상황에서 장애 시나리오를 재현하여, 클라이언트에서 서버 장애에 상관없이 안정적인 접속을 할 수 있는지에 대한 테스트를 진행 하였다.

WAS 와 Zookeeper 메인 서버와 연결하는 방식은 배제 하였으며, 테스트용으로 세 개의 소켓 서버를 생성하여 이용하는 네이밍 서비스 방식을 사용하였다.

□ 테스트 시나리오

[표 V-6. 테스트 시나리오]

시나리오	상태
1.메인서버 Zookeeper 생성	
2.소켓 서버 3개 등록	1번,2번,3번 서버가 주키퍼에 등록됨
3.메인서버에서 서버상태 체크	1초마다 확인
4.클라이언트에서 메인서버 접속	정상적인 서버(소켓 서버1)에 접속된 상태
5.임의의 장애 발생	소켓 서버 1을 강제종료
6. 클라이언트 정상 동작 확인	서버2에 정상적으로 접속

□ 테스트 결과

[표 V-7. 테스트 결과]

시나리오	상태	세션	기타
1.메인서버 Zookeeper 생성			
2.소켓 서버 3개 등록	1번,2번,3번		
3.메인서버에서 서버상태 체크	각 서버별로 1초마다 확인		
4.클라이언트에서 메인서버 접속	정상적인 서버 (소켓 서버1)에 접속된 상태	세션1 연결됨	
5.임의의 장애 발생	소켓 서버 1을 강제종료	세션1 timeout 발생/세션1은 삭제됨	30초 카운트 후 서버1 삭제
6. 클라이언트 정상 동작 확인	서버2에 정상적으로 접속	세션2 연결	

각 서버의 상태 체크는 1초마다, 클라이언트와 메인 서버간의 세션 타임아웃은 30초로 설정.

클라이언트가 접속한 1번 서버에 임의의 장애를 발생시켰을 때
1번 서버에 연결 되면서 생성 되었던 세션은 타임아웃(30초)이 걸리면서 이벤트를 발생 시켰다.

이벤트가 발생되면 Zookeeper 메인서버목록에서 1번 서버를 제거하고 세션 1을 종료하였다. 클라이언트는 Zookeeper 메인서버에서 장애가 나지 않은 서버의 리스트 중 하나(서버2)를 받아와서 정상적으로 접속되는 것을 확인하였다.

Ⅵ. 종합

□ Zookeeper 기능 테스트 수행 결과 공개SW로 구성된 Stack 상에서 각 기능 시나리오 수행 시 치명적 오류 또는 심각한 장애가 발생하지 않았으며, Stack을 구성하는 각 공개SW가 유기적으로 동작함을 확인하였다.

□ Zookeeper 성능 테스트 수행은 Java로 로드밸런싱 소스를 만들어 테스트를 진행하고 세션 타임아웃 시간과 서버에 등록 등의 시간을 임의로 설정하였다.

□ 실제 운영상황에서는 해당 장비에 Zookeeper 및 WAS 연결, 멀티 서버 구현을 하고 부하 및 장애가 발생됐을 때 클라이언트가 정상 접속을 하는지 테스트가 필요할 것으로 판단된다.

※ 참고 자료

[1] http://zookeeper.apache.org/

[2] https://cwiki.apache.org/confluence/display/ZOOKEEPER/Index

[3] 클라우드 컴퓨팅 구현 기술(에이콘 출판사)

[4] Hadoop 완벽가이드(한빛미디어)

[5] http://www.centos.org/

[6] http://www.ubuntu.com/