한국컴퓨터정보학회 동계학술대회 논문집 제23권 제1호 (2015. 1)

대규모 분산 환경을 위한 서비스 디스커버리 기술

김유진*, 윤희용°
*°성균관대학교 정보통신대학
e-mail: ejggle@gmail.com*, youn7147@skku.edu°

Service Discovery Technology for Large-scale Distributed Environment

Eujin Kim*, Hee Yong Youn°
*°College of Information & Communication Engineering, Sungkyunkwan University

● 요 약 ●

서비스 디스커버리는 대부분의 분산시스템 및 서비스 지향 아키텍처의 핵심 구성요소다. 실시간 시스템 기반에서 서비스 위치는 자주 변경될 수 있는데, 이 때 서비스 중단 문제가 발생할 수 있다. 이를 방지하기 위해서 동적인 서비스 등록과 서비스 디스커버리 기법이 매우 중요하다. 본 논문은 서비스 중단 문제를 해결할 수 있는 몇 가지 오픈 소스 솔루션들을 소개한다. 각 솔루션들은 레지스트리 타입에 따라 크게 범용 레지스트리와 단일 목적용 레지스트리로 나눌 수 있다. 각 솔루션들의 기능을 서로 비교함으로써 사용자로 하여금 자신의 요구사항에 적합한 솔루션을 선택하는데 도움이 되고자 한다.

키워드: 서비스 디스커버리(Service Discovery), 서비스 등록(Service Registration)

I. 서 론

웹 2.0의 비즈니스 소프트웨어 및 하드웨어 자산, e-비지니스 또는 소셜 소프트웨어 응용프로그램 등을 포함하는 웹 서비스의 지속적인 확산은 오늘날의 사회 경제에서 상호 작용 방식의 더 혁명적인 변화를 가져왔다. 서비스는 공급자와 클라이언트(Client) 사이의 값(Value)이 교환되는 동작, 수행, 또는 약속의 일종으로 정의할 수 있다. 즉, 서비스는 모든 관련 당사자들의 값을 만들고 캡처(Capture)하는 공급자 클라이언트 상호 작용(Provider-Client Interaction)이다.

서비스 디스커버리(Service Discovery)는 기능적 또는 비 기능적 시맨틱(Semantics)의 설명을 기반으로 주어진 요청에 따라 관련되어 존재하는 서비스를 찾는 과정이다. 서비스 디스커버리에 대한 접근 방식은 지원하는 서비스 기술 언어(Service Description), 검색의 구조, 그리고 서비스 선택의 활용 수단에 따라 다르다.

서비스 디스커버리는 대부분의 분산 시스템(Distributed Systems)과 서비스 지향 아키텍처(Service Oriented Architectures)의 핵심 구성요소다. 실시간 시스템 기반에서 서비스 위치는 자주 변경될 수 있는데, 여기서 서비스 중단 문제가 발생한다. 이를 방지하기 위해서 동적인 서비스 등록(Service Registration) 및 서비스 디스커버리 기법이 매우 중요하다. 이 문제를 해결하기 위해 활발히 연구되고 있는 추세이다.

본 논문은 서비스 등록과 서비스 디스커버리의 개념을 소개하고, 위에서 언급한 서비스 중단 문제를 해결할 수 있는 몇 가지 오픈 소스(Open Source) 솔루션들을 소개한다. 각 솔루션들은 레지스트리 타입에 따라 크게 범용 레지스트리와 단일 목적용 레지스트리로 나눌 수 있다. 솔루션들의 기능을 서로 비교함으로써 사용자로 하여금 자신의 요구사항에 적합한 솔루션을 선택하는데 도움이 되고자 한다.

II. 관련 연구

1. 서비스 등록 및 서비스 디스커버리

서비스 위치 탐지의 문제에는 서비스 등록 및 서비스 디스커버리의 두 가지 측면이 있다. 먼저 서비스 등록은 중앙 레지스트리에 위치를 등록하는 서비스 프로세스다. 보통 호스트와 포트를 등록하고, 때때로 자격 증명, 프로토콜, 버전 번호 및 환경 세부사항을 등록하기도 한다. 그리고 서비스 디스커버리는 서비스 위치를 알기 위해 중앙 레지스트리에 요청하는 클라이언트 애플리케이션 프로세스를 뜻한다.

지원하는 서비스 기술 언어(Service Description Language), 서비스 프로세스 처리 방법, 그리고 검색에 사용되는 서비스 선택의 방법에 따라 다르게 수행될 수 있다. 이러한 모든 서비스 등록 및 서비스 디스커버리 솔루션은 다른 개발 및 운영 측면에서 모니터링(Monitoring), 부하 균형(Load Balancing), 런타임 종속성, 가용성 등을 고려해야 한다[1][2].

2. 서비스 기술 언어

웹 서비스는 무엇을 하고, 또 실제로 어떻게 작동하는 지에 관해서

표현된다. 웹 서비스의 기능적인 측면은 서비스 프로파일(Service Profile)과 서비스 프로세스 모델로 나누어 접근할 수 있다.

서비스 프로파일은 입출력 매개변수(I/O Parameters)의 관점에서 서비스의 특성 및 서비스 사양을 기술한다. 즉, 서비스 실행의 전제 조건과 결과를 표현한다. 뿐만 아니라 출처, 이름, 업종, 가격, 배달 제한 및 품질에 대한 정보와 같은 비 가능적 서비스 시멘틱(Sementic) 또한 기술한다. 대표적인 프로파일은 XML 기반의 웹서비스 기술 언어 WSDL, SML, USDL, WADL, HTML 마이크로 포멧 hREST (시맨틱 REST) 등이 있다. 또한 레스트풀(RESTful) 서비스의 텍스트 문서와 온톨로지(Ontology) 기반의 서비스 기술 언어 OWL-S, WSML, SAWSDL, SA-REST 및 Linked USDL도 예로 들 수 있다[4].

서비스 프로세스 모델은 내부 제어와 데이터 흐름의 관점에서 서비스의 작동을 기술한다. 서비스 프로세스 모델은 표준 작업 흐름 오퍼레이터(Operator)를 사용하는 OWL-S, WSML, USDL 등에서 구현한다. 반면 파이 미적분(Pi-calculus) 및 페트리 네트(Petri-nets)와 같은 프로세스 대수 언어를 채택하여 구현하기도 한다[3].

III. 본 론

1. 범용 레지스트리

Zookeeper, Doozer 및 Etcd 레지스트리 모두 매우 일관성 있는 프로토콜을 사용하고, 실제로 데이터 저장소가 일관된 범용이다. 여기서는 이 세 레지스트리를 서비스 레지스트리로서 다룬다. 하지만 이 외에도 대표 선출에 도움이 되는 조정 서비스(Coordination Service)에 사용되거나, 클라이언트의 분산된 세트를 잠그는 것에 활용되기도 한다.

1.1 Zookeeper

Zookeeper는 구성 정보 유지, 명명, 분산된 동기화(Synchronization) 제공 및 그룹 서비스 제공에 대한 집중 서비스이다. Zookeeper는 자바로 작성되었고, 변경사항을 조정하기 위해 Zab 프로토콜을 사용한다. Zookeeper는 일반적으로 앙상블(Ensemble)에서 세 개, 다섯 개, 또는 일곱 개의 멤버로 실행된다. 앙상블에 엑세스하기 위해 클라이언트는 특정 언어 바인딩(Language Specific Binding)을 사용한다. 보통 엑세스는 클라이언트 애플리케이션과 함께 서비스에 포함된다.

서비스 등록은 명칭 공간(Namespace)에서 임시 노드(Node)로 구현한다. 클라이언트가 연결되는 동안에는 임시 노드만 존재해서, 시작 후에 백엔드(Backend) 서비스는 위치 정보에 자체적으로 등록한다. 작업이 실패하거나 연결이 끊기면 노드는 트리에서 사라진다.

서비스 디스커버리는 서비스에 대한 명칭 공간을 열거하고 지켜봄으로써 구현된다. 클라이언트는 현재 등록된 서비스는 물론, 서비스를 사용할 수 없게 되거나 서비스가 새로 등록될 때의 알림을 모두 받는다. 부하 균형이나 시스템 대체 작동(Failover) 또한 클라이언트가 처리한다.

Zookeeper는 매우 일관성 있는 시스템이다. 따라서 분할(Partition)이 발생했을 때, 분할 시에 기능이 정상적이어도 문제가 생긴다. 시스템의 일부에서 등록 할 수 없고, 기존 등록도 찾을 수 없다. 특히, 비 쿼럼(Quorum) 측면에서 읽기와 쓰기는 오류를 반환한다[5].

1.2 Doozer

Doozer는 일관성 있고 분산된 데이터 저장소이다. Go로 작성되고, 매우 일관성 있으며 일치를 유지하기 위해 Paxos를 사용한다. Doozer는 일반적으로 클러스터(Cluster)에서 세 개, 다섯 개, 일곱 개의 노드로 실행된다. 클라이언트는 특정 언어 바인딩을 사용하고, Zookeeper와 마찬가지로 클라이언트와 서비스에 포함된다.

하지만 Doozer는 임시 노드의 개념이 없기 때문에, Zookeeper에 비해 서비스 등록 과정이 복잡하다. Doozer에서 서비스는 경로에 스스로 등록할 수는 있지만, 서비스가 사용할 수 없게 되더라도 자동으로 제거되지 않는다. 이 문제점을 해결하기 위한 방법은 여러 가지 있다. 그 중 하나는 다음과 같다. 먼저 레지스트레이션 프로세스에 타임 스템프(Time Stamp)와 하트비트(Heartbeat) 메커니즘(Mechanism)을 추가한다. 그리고 디스커버리 프로세스 동안 만료된 엔트리를 처리한다.

서비스 디스커버리는 Zookeeper와 비슷하다. 경로에 있는 모든 엔트리를 나열하고 해당 경로로의 변경을 기다린다. 레지스트레이션에 타임 스템프와 하트 비트 메커니즘을 사용하는 경우, 검색 시 만료된 엔트리는 무시되거나 삭제된다.

Zookeeper와 같이, Doozer 또한 매우 일관성 있는 시스템이며 분할이 발생했을 때 동일한 결과를 갖는다.

1.3 Etcd

Etcd는 공유된 구성에 대한 키 값 저장소이며, 가용성 또한 높다. Etcd는 Zookeeper와 Doozer의 영향을 받았으며, Doozer와 마찬가지로 Go로 작성되었다. 또한 일치를 위해 Raft를 사용하고, API 기반의 HTTP+JSON을 갖는다. Zookeeper 및 Doozer와 유사하게 보통 세 개, 다섯 개, 또는 일곱 개의 노드로 실행된다. 클라이언트는 특정 언어 바인딩을 사용하거나, HTTP 클라이언트를 이용하여 구현한다.

서비스 등록은 하트비트 메커니즘과 마찬가지로 TTL(Time to Live) 값을 이용한다. 서비스가 TTL 값 업데이트 를 실패하면 Etcd가 만료된다. 서비스를 사용할 수 없게 되면, 클라이언트는 연결 실패를 처리하고 다른 서비스 사용을 시도한다.

서비스 디스커버리는 디렉토리(Directory)의 키를 나열하고, 디렉토리에서 변경을 기다린다. HTTP 기반 API 때문에, 클라이언트 애플리케이션은 Etcd 클러스터로 긴 폴링(Polling) 연결을 유지한다.

Raft를 사용하는 Etcd 또한 강력하게 일관성 있는 시스템이다. Raft는 선출된 대표를 필요로 하고, 모든 클라이언트 요청이 이 대표에 의해 처리된다. 하지만 Etcd는 비공식적으로 일관성 있는 매개변수를 사용하는 비 대표 읽기를 지원한다. 여기에서 매개변수는 읽을 시 가용성을 향상시킨다. 분할이 발생했을 때, 쓰기는 대표에 의한 처리되어야 하며 쓰기가 실패할 수도 있다.

2. 단일 목적 레지스트리

Serf, SmartStack, SkyDNS2 및 Eureka와 같은 단일 목적 레지스

트리는 서비스 등록과 서비스 디스커버리에 맞게 구체적으로 조정되었다. Zookeeper, Doozer 및 Etcd과 같은 범용 레지스트리는 분산된 구성에 이용할 수 있는 반면, 단일 목적 레지스트리는 그러한 기능이 없다.

2.1 SmartStack

SmartStack은 사용자 도구인 Nerve와 Synapse의 결합이다. Nerve와 Synapse는 서비스 등록과 서비스 디스버커리를 처리하는 HAProxy와 Zookeeper에 영향력을 미친다. Nerve와 Synapse 모두 Ruby로 작성된다.

Nerve는 사이드킥(sidekick) 스타일 프로세스다. 사이드킥 스타일 프로세스는 애플리케이션 서비스에서 별개의 프로세스로 실행한다. Nerve는 Zookeeper에 서비스를 등록한다. 애플리케이션은 HTTP 서비스에 대한 /heath 종점을 드러낸다. 이를 Nerve가 계속해서 모니터한다. 이용 가능한 서비스가 제공되면 Zookeeper에 등록된다.

Synapse 또한 Nerve와 같은 사이드킥 스타일 프로세스이다. 서비스와 함께 별도의 프로세스로 실행된다. Synapse는 서비스 디스커버리를 담당한다. 최근 등록된 서비스에 대하여 Zookeeper를 요청하고, 로컬로 실행하는 HAProxy를 변경한다. 다른 서비스에 접근할 필요가 있는 호스트의 모든 클라이언트는 로컬 HAProxy를 엑세스한다. 여기서 HAProxy는 이용 가능한 서비스에 요청을 보낸다. Synapse의 디자인은 클라이언트 측면의 부하균형 또는 대체작동(Failover)을 실행하지 않는다. 또한 Zookeeper나 언어 바인딩을 필요로 하지 않기 때문에 서비스 구현을 간단히 할 수 있다.

SmartStack은 Zookeeper에 의존하기 때문에 일부 레지스트레이션 및 디스커버리가 분할하는 동안 실패가 발생할 수 있다. 서비스가 분할하기 전에 최소 한 번 다른 서비스를 발견할 수 있으면, 분할 후에 서비스의 스냅샷(Snapshot)을 갖고 분할하는 동안 계속 작동할 수 있다. 이러한 측면은 전체 시스템의 가용성 및 안정성을 향상시킨다.

2.2 Serf

Serf는 서비스 검색 및 결합을 위한 분산 솔루션이다. Serf 또한 Go로 작성되었지만, 가십(Gossip) 기반 프로토콜, SWIM, 장애 감지 및 사용자 정의 이벤트 전파를 이용한다는 점에서 차이가 있다. SWIM은 기존 하트비트 프로토콜의 비 확장성을 해결하기 위해 설계 되었다. Serf는 모든 호스트에 설치되어 있는 단일 이진(Single Binary)으로 구성된다. Serf는 클러스터를 결합하거나 생성하는 곳에서 에이전트(Agent)로서 실행되거나, 클러스터의 멤버들을 발견할 수 있는 곳에서 클라이언트로서 실행될 수 있다.

Serf 에이전트는 기존 클러스터를 결합하여 서비스 등록을 한다. 에이전트는 호스트의 역할, IP 및 포트 등을 식별할 수 있는 커스텀 태그(Custom Tag)와 함께 시작된다. 한 번 클러스터에 결합되면, 다른 멤버가 여기의 호스트 및 메타데이터(Metadata)를 볼 수 있다.

서비스 디스커버리의 경우, Serf는 클러스터의 현재 멤버를 반환하는 멤버 명령으로 실행된다. 멤버 출력을 이용해 서비스의 모든 호스트를 검색할 수 있다. 여기서 서비스는 실행중인 에이전트 태그를 기반으로 한다.

Serf는 다른 솔루션들과 비교하면 비교적 새로운 솔루션에 속하며, 빠르게 발전 중이다. 여기서 소개한 여러 프로젝트 중 유일하게 중앙 레지스트리 아키텍처 스타일을 갖지 않는다. Serf는 가십 기반 프로토콜인 비동기식(Asynchronous)을 이용하기 때문에 일관성이 약하다.

표 1. 서비스 디스커버리 오픈 소스 솔루션 비교
Table 1. Comparison of Service Discovery Open Source Solutions

	Zook-eeper	Doozer	Etcd	Smart Stack	Serf
레지스트리 종류	범용	범용	범용	단일 목적용	단일 목적용
개발	Apache	Heroku	CoreOS	Airbnb	HashiCorp
개발 년도	2010	2010	2013	2013	2013
언어	Java	Go	Go	Ruby	Go
통합	Client Binding	Client Binding	Client Binding /HTTP	Sidekick (Nerve/Synapse)	Local CLI

IV. 결 론

본 논문은 오픈 소스 기반의 서비스 중단 문제의 해결방안에 대해 살펴보고 각각의 기능들을 서로 비교해 보았다. 이 내용들을 바탕으로 사용자들은 자신의 요구사항에 적합한 솔루션을 선택할 수 있을 것이라고 기대한다.

ACKNOWLEDGMENT

본 연구는 BK21+사업, 한국연구재단 기초연구사업 (2012R1A1A2040257), (2013R1A1A2060398), 삼성전자(S-2014-0700-000), 미래창조과학부 및 정보통신기술연구진흥센터의 정보통신·방송 연구개발사업(1391105003)의 일환으로 수행하였음.

참고문헌

[1] Marco Crasso, Alejandro Zunino and Marcelo Campo, “A Survey of Approaches to Web Service Discovery in Service-Oriented Architectures,” J Database Management, IGI Global. 2011.

[2] John Garofalakis, Yannis Panagis, Evangelos Sakkopoulos and Athanasios Tsakalidis, “Contemporary web service discovery mechanisms,” J Web Engineering, Rinton Press, 2006.

[3] Matthias Klusch, “Service Discovery,” Reda Alhajj and Jon Rokne (Eds.), Encyclopedia of Social Network and Mining (ESNAM), Springer, 2014.

[4] Afaf Madani Mohamed and Robert M. Colomb, “Study Survey of Service discovery using perdurant ontology,” ICCEEE, 2013.

[5] Lars George, “HBase: The Definitive Guide,” O'Reilly Media, Inc., Sep. 2011.