고성능 시스템을 위한 세션 상태 저장소 복제

이창엽, 최창열, 김성수
아주대학교 정보통신전문대학원
e-mail : {cylee, clchoi, sskim}@ajou.ac.kr

Session State Repository Replication for High Performance System

Chang-Yup Lee, Chang-Yeol Choi, Sungsoo Kim
Graduate School of Information and Communication, Ajou University

요 약

시스템은 한번에 많은 양의 작업을 처리할 때 과부하로 인해 종종 서비스 제공이 불가능하게 된다. 세션 상태의 이용은 시스템 관리를 용이하게 하여 이런 문제점의 해결에 도움을 줄 수 있다. 하지만 세션 상태의 저장소를 관리하기 위한 기존의 SSM(Session State Manager)이 한번에 처리할 수 있는 서비스의 양에는 한계가 있다. 따라서 본 논문에서는 세션 상태 저장소의 복제를 통해 시스템의 성능을 높이고 서비스의 응답 시간을 줄이기 위한 메커니즘을 제안한다.

1. 서론

모든 시스템은 한번에 처리할 수 있는 서비스의 양에 한계를 가지고 있기 때문에 그 한계를 넘는 수의 서비스 요청을 처리할 때 과부하로 인해 종종 서비스 제공이 불가능하게 된다. 많은 수의 사용자가 동시에 하나의 웹 사이트에 접속했을 때 서버가 다운되는 현상을 예로 들 수 있으며 이를 해결하기 위해선 많은 수의 서비스 요청을 빨리 처리할 수 있는 메커니즘이 필요하다. 세션 상태를 이용하면 서비스 처리 속도를 높일 수 있기 때문에 시스템 관리가 용이해져서 위와 같은 문제 해결에 도움을 준다 [1]. 쇼핑 카트나 스케줄러처럼 현재 상황에 대한 정보를 저장하는 것을 세션 상태의 예로 들 수 있다. 세션 상태는 일정 시간 동안 존재하다가 그 시간이 지나면 소멸되는데, 존재하는 시간 동안 지속성이 보장되고 높은 가용성을 제공한다 [2]. SSM[3]은 이러한 세션 상태의 저장소를 제공함으로써 세션 상태 관리를 용이하게 하지만 한꺼번에 자신이 다 처리할 수 없는 양의 요청이 들어오면 시스템의 안정성을 위해 서비스 요청을 거부한다. 그렇기 때문에 SSM 은 많은 양의 서비스를 처리하는 시스템에 사용하기에 한계가 있다. 따라서 본 논문에서는 세션 상태의 저장소를 증가시켜서 대규모의 서비스 요청을 처리함으로써 시스템의 성능을 높이고 서비스의 응답 시간을 줄일 수 있는 방법을 제안하고자 한다.

이 논문은 2005 년도 두뇌한국 21 사업에 의하여 지원되었음.
본 연구는 정보통신부 21 세기프론티어연구개발사업의 일환으로 추진되고 있는 유비쿼터스컴퓨팅및네트워크원천기반기술개발사업의 지원에 의한 것임.

2. 관련연구

Byzantine resilient 시스템 [4]은 최소한의 프로세서들과 연결들을 사용하여 시스템 내의 모든 프로세서들을 동기화시킨다. f 개의 오류를 견디기 위한 프로토콜인 f-Byzantine resilient 시스템은 적어도 $(3f + 1)$개의 프로세서가 있어야 한다. 하지만 본 논문에서 소개하는 시스템은 모든 프로세서들을 동기화시킬 필요가 없기 때문에 Byzantine resilient 시스템보다 적은 수의 프로세서를 필요로 한다.

AIMD(Additive Increase Multiplicative Decrease) 메커니즘 [5]은 TCP 의 혼잡제어 기술과 유사하다. AIMD 는 시스템의 작업 처리 속도에 맞춰서 시스템에 전송되는 요청의 양을 조절하는 기술로서, Tahoe 알고리즘과 Reno 알고리즘 등이 있다. Tahoe 알고리즘은 아래에 기술되어 있다. W 는 현재 윈도우 크기이고, W_t 는 임계값이며, W_M 은 최대 윈도우 크기라고 할 때,

- 처음의 윈도우 크기는 1 이다.
- 바로 전 윈도우로부터 모든 응답을 받았을 때,

- 만약 $W < W_t$ 이면, 윈도우 크기를 두 배로 증가시킨다.
- 만약 $W >= W_t$ 이면, 윈도우 크기를 1 증가시킨다.

● 시간 초과 신호를 받으면 윈도우 크기를 1 로 줄이고, W_t 를 반으로 줄인다.

Reno 알고리즘은 Tahoe 알고리즘에서 아래의 규칙이 추가되었다.

● 손실 신호를 받으면 W_t 를 반으로 줄이고 현재 윈도우 크기를 W_t 로 정한다.

3. DSSR(Dynamic Session State Repository)의 구조와 메커니즘

3.1 DSSR 의 구조

DSSR 은 세션 상태의 저장소이다. 하지만 기존에 소개된 SSM 과는 달리 로드의 양에 따라서 저장소의 수를 증가시키거나 감소시킬 수 있다. DSSR 은 스텁(stub)과 브릭(brick), 그리고 브릭 관리자(brick manager)로 이루어져 있다. 어플리케이션 서버는 세션 상태를 읽거나 저장할 때 스텁을 이용한다. 스텁은 어플리케이션 서버로부터 받은 읽기 또는 쓰기 요청을 브릭으로 전송한다. 브릭은 소프트웨어 컴포넌트로서 세션 상태를 해시 테이블에 저장한다. 브릭 관리자는 전송되는 로드의 양에 따라서 브릭을 추가하거나 제거하는 역할을 한다. 하나의 브릭 관리자와 하나 이상의 브릭은 하나의 노드 안에 존재하는데, 노드는 물리적인 기계이므로 그 수가 고정되어 있다. 가까운 곳에 위치한 노드들은 SAN(system area network)에 의해 연결되며, 이로 인해 높은 대역폭을 제공하여 지연 시간을 줄일 수 있다. DSSR 에서는 안정적인 통신을 위해 TCP/IP 프로토콜을 사용하며, 그림 1 은 DSSR 의 구조를 보여준다.

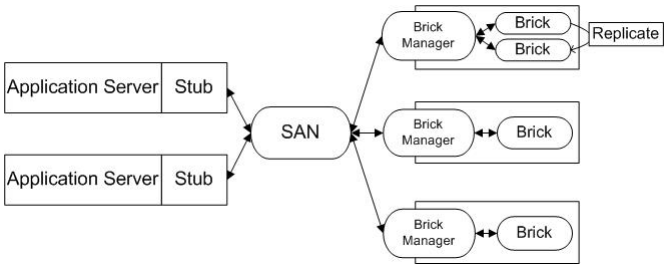

그림 1. DSSR 의 구조

사용자가 서비스를 요청하면 어플리케이션 서버는 스텁에게 사용자의 세션 상태를 읽거나 저장하는 것을 요구한다. 스텁이 세션 상태를 브릭으로부터 읽어오거나 브릭에 저장하라는 요청을 보내면 브릭은 요청을 수행한 다음에 그에 따른 응답을 스텁으로 보낸다. 읽기와 쓰기 요청의 기본 철학은 "많은 수의 요청을 보내고 적은 수의 응답을 기다린다." 이며, 이렇게 함으로써 어느 정도의 오류를 견딜 수 있다. 쓰기 요청시, f_B 를 브릭의 서비스 중지 또는 전송 실패로 인해 생기는 오류이고, f_R 을 쓰기 요청에 대한 응답 중에서 잘못된 값을 가지는 응답의 수라고 하면, 스텁에서는 요청에 대한 응답에 대해서 다수결에 따라 올바른 값을 선택하기 때문에 f_R 개의 오류를 견디기 위해서 적어도 2 f_R + 1 개의 응답을 받아야 한다. 또한 전송한 요청 중에서 f_B 를 견디기 위해서는 추가로 f_B 개의 요청이 더 필요하다. 따라서 쓰기 요청시, 스텁은 적어도 $2f_R + f_B + 1$ 개의 브릭에 요청을 보내야 하고 $2f_R + 1$ 개 이상의 응답을 받아야 한다. 이것은 곧 시스템 내에 $2f_R + f_B + 1$ 개 이상의 브릭이 존재해야 하고, 만약 n 개의 노드가 있으면 하나의 노드에는 $\left\lceil \frac{2f_R + f_B + 1}{n} \right\rceil$ 개의 브릭이 존재해야 된다는 것을 의미한다. 그림 2 는 개략적인 쓰기 요청 수행 과정을 보여준다.

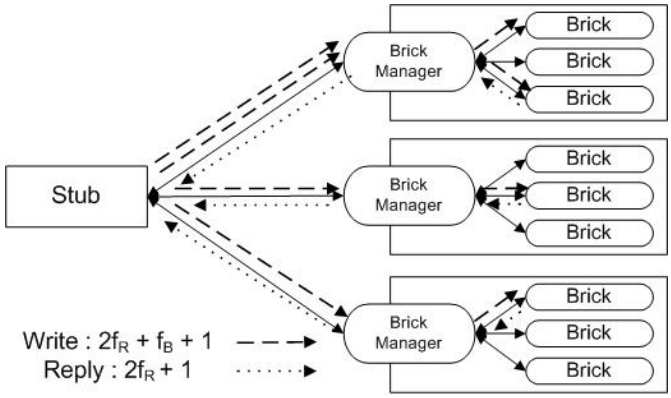

그림 2. 쓰기 요청 수행 과정

읽기 요청은 가장 최근에 저장된 값을 불러와야 하기 때문에 쓰기 요청에 대한 응답 중 올바른 값을 가진 브릭에 대한 정보가 메타데이터로서 사용자에게 전달된다. 사용자는 읽기 요청시 그 메타데이터를 사용하여 가장 최근에 저장된 값을 얻을 수 있다. 스텁은 브릭에게 f_R 개의 읽기 요청을 보내고 하나의 응답만을 기다린다.

그림 3 은 DSSR 의 전체적인 메커니즘을 보여준다. 그림 3 을 쇼핑몰 사이트에서의 장바구니를 예로 들어 설명하겠다. 한 명의 사용자가 쇼핑몰 사이트에 로그인을 하면 그 사용자에 대한 정보를 데이터베이스 서버에서 불러온다. 사용자가 한 상품을 장바구니에 추가하면 쓰기 요청이 스텁에 전달되어 브릭 안의 해시 테이블에 저장된다. 쓰기 요청에는 네 가지 중요한 정보가 담겨있는데, 해시 키, 세션 상태 객체, 세션 상태의 수명, 그리고 현재의 윈도우 크기가 그것이다. 세션 상태는 객체 형태로 브릭에 저장되며, 브릭 안의 해시 테이블에 저장되기 위해 해시 키가 필요하다. 세션 상태는 정해진 수명이 다하면 자동으로 소멸되는데 사용자의 수명 갱신 메시지에 의해 수명이 늘어날 수 있다. 현재 윈도우의 크기는 브릭 관리자가 브릭의 수를 조정하기 위해 사용한다. 사용자가 장바구니 안에 있는 상품들을 보기를 원하면 읽기 요청이 전달되어 가장 최근에 저장된 값을 읽어온다. 위에서 언급한

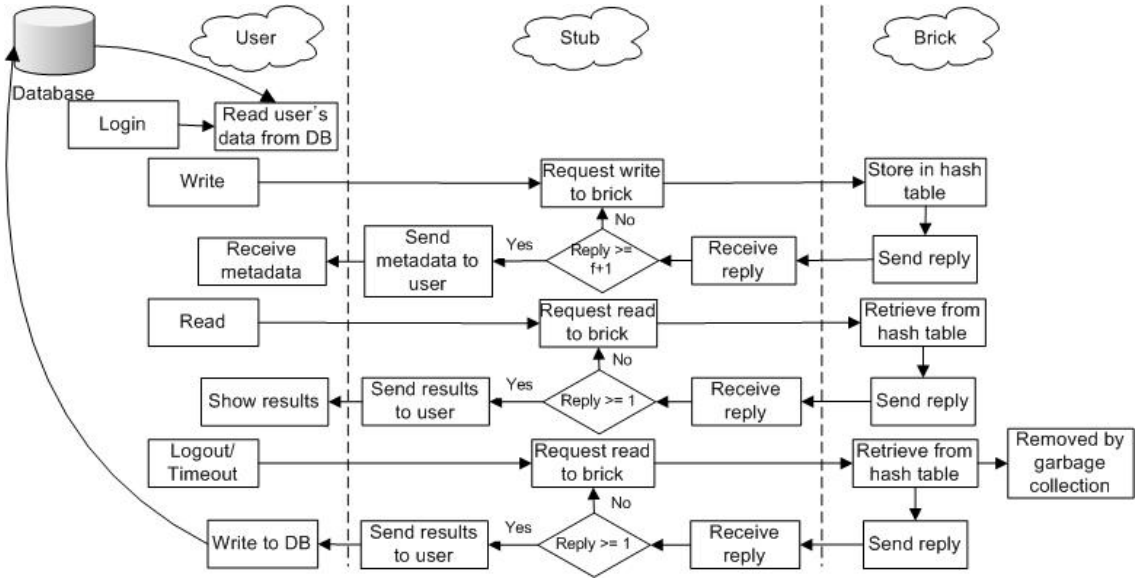

그림 3. DSSR 의 전체적인 메커니즘

것처럼 스텁은 쓰기 요청을 $2f_R + f_B + 1$ 개 이상의 브릭에 전달하여 $2f_R + 1$ 개 이상의 응답을 받아야 하고, f_R 개의 읽기 요청을 보내서 하나 이상의 응답을 받아야 한다. 만약 일정한 시간 t 동안 필요한 만큼의 응답을 받지 못하면 같은 요청을 다시 보내는데, t 는 RTT(round trip time)로 정해진다. 사용자가 로그아웃하거나 세션 상태의 수명이 다하면 사용자가 장바구니에 담은 상품들은 데이터베이스 서버에 저장된다. 세션 상태의 수명이 다하면 가베지 수집(garbage collection)에 의해 브릭 안의 해시 테이블에서 제거된다.

3.2 DSSR 의 브릭 관리 메커니즘

스텁은 윈도우의 크기를 관리하여 브릭으로 보내는 요청의 수를 조정한다. 윈도우는 스텁이 브릭으로 보낼 수 있는 최대 요청수로, 브릭으로 전송한 요청 중 응답을 받지 않은 요청의 수가 윈도우 크기와 같아지면 스텁은 더 이상 브릭에 요청을 보내지 않는다. AIMD 알고리즘은 윈도우 크기를 조절하는 알고리즘이다. 스텁이 요청에 대한 정상적인 응답을 받았다면 브릭이 좀 더 많은 요청을 처리할 수 있다는 것을 의미하기 때문에 윈도우 크기를 늘려서 더 많은 요청을 보낸다. 반대로 일정 시간 t 가 지날 때까지 응답이 오지 않으면 브릭이 현재의 요청들을 다 처리할 수 없다는 것을 의미하기 때문에 윈도우 크기를 줄인다. 이렇게 함으로써 브릭이 과부하로 인해 서비스 불능 상태가 되는 것을 막는다. DSSR 에서는 윈도우 크기 조정 알고리즘으로 Reno 알고리즘을 사용한다. Reno 알고리즘은 데이터 손실 신호를 받으면 임계값을 현재 임계값의 반으로 줄이고 현재 윈도우 크기를 그 임계값으로 줄인다. 처음의 임계값은 최적의 윈도우 크기로 정해지는데, 최적의 윈도우 크기는 b 가 네트워크의 대역폭이고 τ 가 RTT 일 때, bτ 로 정하였다 [6].

브릭 관리자는 윈도우의 크기를 이용하여 브릭의 수를 관리하는데, 현재 윈도우의 크기는 쓰기 요청에 저장된 정보를 통해 얻어온다. 윈도우의 크기가 최소라는 것은 브릭에서 요청들을 다 처리할 수 없다는 것을 의미한다. 이 때 윈도우의 크기를 줄이는 것으로 시스템의 안정성을 높일 수 있지만 처리해야 할 요청들을 거부하기 때문에 서비스의 응답 시간은 길어지게 된다. 그렇기 때문에 더 많은 양의 요청을 처리하기 위해 브릭의 수를 늘려야 한다. 윈도우의 크기가 최대라는 것은 현재 전송되고 있는 요청의 수가 많지 않다는 것을 의미하므로 브릭의 수를 줄여도 된다. DSSR 은 f_B 개의 손실을 견딜 수 있게 때문에 f_B 보다 적은 수의 브릭이 응답하지 않아도 정상적으로 작동할 수 있다. 하지만 f_B 보다 많은 수의 브릭이 응답하지 않으면 치명적인 결함이 나타날 수 있다. 이를 방지하기 위해 브릭의 수를 줄일 때는 일정한 시간 간격을 두고 하나씩 줄여야 한다. 이 시간 간격 역시 하나의 요청에 대한 RTT 로 정할 수 있다. 브릭 수 조정 규칙은 Reno 알고리즘과 유사하게 현재 브릭의 수를 임계값과 비교하여 증가량에 차이를 둔다. 최소 윈도우 크기는 1 이며, B 는 현재의 브릭 수라고 하고, Bt 는 임계값이라고 할 때,

- 하나의 노드 안에서 처음의 브릭 수는 $\left\lceil \frac{2f_R + f_B + 1}{n} \right\rceil$ 이다.
- 윈도우 크기가 최소일 때,
  - $B < Bt$ 이면 브릭의 수를 두 배로 한다.
  - $B >= Bt$ 이면 브릭의 수를 1 증가시킨다.
- 윈도우 크기가 최대일 때,
  - 브릭 수를 1 감소시킨다.

브릭의 임계값 Bt 는 노드의 성능에 따라 결정된다.

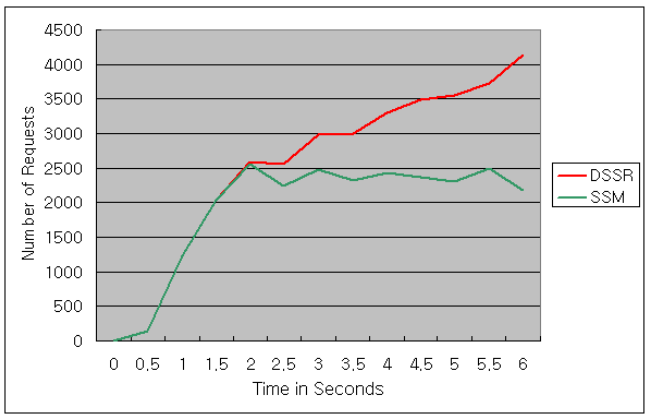

그림 4. 성공적으로 처리된 요청의 수

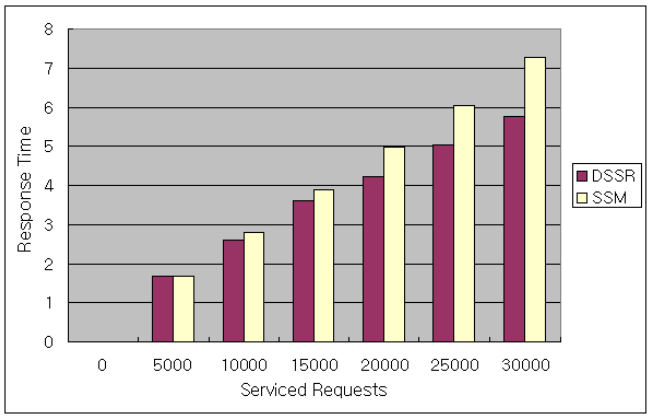

그림 5. 일정한 수의 요청 처리에 필요한 응답 시간

4. 성능 분석

이 장에서는 DSSR 과 SSM 의 성능과 응답 시간을 비교하였다. 성능은 단위 시간 내에 처리된 요청의 양이고, 응답 시간은 일정한 양의 요청을 처리하는데 걸린 시간이다. 노드의 수는 네 개이고 처음의 브릭 수는 각 노드에 하나씩으로 하여 역시 네 개로 정했다. 네트워크의 대역폭은 요청들을 모두 통과시킬 수 있을 만큼 크다고 가정하였고 타임아웃은 100ms 로 정하였다. 시뮬레이션의 결과는 그림 4 와 그림 5 에서 볼 수 있는데, 그림 4 는 두 시스템의 성능을 비교하였고 그림 5 는 응답 시간의 차이를 비교하였다. 그림 4 에서 x 축은 시간을 나타내며 y 축은 시간에 따라 성공적으로 처리된 요청의 수를 나타낸다. 처음에는 전송되는 요청의 수가 적기 때문에 두 시스템의 요청 처리 속도가 비슷하지만 시간이 갈수록 DSSR 이 더 많은 요청을 처리하는 것을 볼 수 있다. 그 이유는 현재 시스템이 가지고 있는 브릭으로 처리할 수 있는 요청의 양에 한계가 왔을 때, SSM 은 윈도우 크기 조절 알고리즘을 통해 전송되는 요청을 거부하지만 DSSR 은 브릭의 수를 증가시킴으로써 더 많은 요청을 수용하기 때문이다. 그림 5 에서 x 축은 처리된 요청의 수이고 y 축은 그 요청을 모두 처리하는데 필요한 응답 시간이다. 응답 시간 역시 적은 양의 요청을 처리할 때에는 두 시스템에 큰 차이가 없지만 많은 양의 요청 처리시 DSSR 에서는 브릭을 증가시키기 때문에 SSM 과의 차이가 점점 커지게 된다.

5. 결론 및 향후 계획

세션 상태를 이용함으로써 시스템 관리에 필요한 비용과 노력을 줄일 수 있다. 세션 상태를 저장하기 위한 시스템으로 SSM 이 소개되었지만 이것은 많은 양의 서비스를 처리하는데 한계가 있다. 따라서 본 논문에서는 이러한 문제를 해결하기 위한 시스템으로 DSSR 을 제안하였다. DSSR 은 보다 많은 양의 서비스를 수용하기 위해 브릭의 수를 늘려서 성능을 높였고, 이로 인해 사용자는 더 짧은 시간에 더 많은 서비스를 제공받을 수 있다. 하지만 DSSR 에는 몇 가지 보완해야 할 점들이 있는데, 먼저 각 노드의 물리적인 성능이 고려되야 한다. 모든 기계들은 성능에 한계가 있기 때문에 하나의 노드 안에서 브릭의 수를 무한정 늘릴 수는 없다. 따라서 노드의 물리적인 성능을 고려하여 최적의 브릭 수를 찾아내는 것이 필요하다. 또한 보다 정확한 서비스를 제공하기 위해 현재의 부하량, RTT 의 정확한 측정과 보다 나은 부하 분배 기술이 필요하다. 이러한 요소들을 고려한다면 DSSR 은 보다 안정적이고 정확한 서비스를 제공할 수 있을 것으로 기대된다.

참고문헌

[1] B. Ling and A. Fox, “A self-tuning, self-protecting, self-healing session state management layer,” Proceedings of the 5th Annual Workshop on Active Middleware Services, pp. 131-137, June 2003.

[2] S. Raman and S. McCanne, “A Model, Analysis, and Protocol Framework for Soft State-based Communication,” Proceedings of the ACM SIGCOMM Conference, pp. 20-25, Sep. 1999.

[3] B. Ling, E. Kiciman, and A. Fox, “Session State: Beyond Soft State,” Proceedings of the 1st Symposium on Networked Systems Design and Implementation, pp. 295-308, Mar. 2004.

[4] D. Pradhan, Fault-tolerant Computer System Design, Prentice-hall, 1996.

[5] C. Liu and E. Modiano, “On the Performance of Additive Increase Multiplicative Decrease (AIMD) Protocols in Hybrid Space-terrestrial Networks,” Proceedings of the Computer Networks, volume 47, pp.661-678, Apr. 2005.

[6] J. Mo, R. La, V. Anantharam and J. Walrand, “Analysis and Comparison of TCP Reno and Vegas,” Proceedings of the INFOCOM Conference, pp. 1556-1563, Mar. 1999.