Vertikale Skalierung für aktiv replizierte Dienste in Cloud-Infrastrukturen

Gerhard Habiger*, Franz J. Hauck*, Johannes Köstler†, Hans P. Reiser†
*Universität Ulm
{gerhard.habiger,franz.hauck}@uni-ulm.de
†Universität Passau
{jk,hr}@sec.uni-passau.de

I. Motivation und Problembeschreibung

Bei aktiver Replikation eines Dienstes wird dieser redundant auf mehreren Rechnern ausgeführt. Eine solche Herangehensweise bringt im Vergleich zu anderen Fehlertoleranzmechanismen wie passive bzw. Primary/Backup-Replikation zwei wesentliche Vorteile mit sich: Zum einen stehen selbst bei einem Fehler in einem Replikat weitere Replikate unmittelbar bereit, so dass im Fehlerfall keine spürbare Beeinträchtigung des Dienstes ensteht. Zum anderen lässt sich neben einfachen Ausfällen (Crash Fault Tolerance – CFT) auch beliebiges Fehlverhalten einzelner Replikate tolerieren (Byzantine Fault Tolerance – BFT). Aktive Replikation hat sich in zahlreichen Anwendungen bewährt, unter anderem für zentrale Koordinierungsdienste von Cloud-Infrastrukturen, wie beispielsweise Chubby [1] und ZooKeeper [2].

Ein zentraler Nachteil von aktiver Replikation ist der dafür notwendige Aufwand. Die redundante Ausführung eines Dienstes nimmt Ressourcen in Anspruch, deren Kosten vor allem dann unnötig hoch ausfallen, wenn das System für den Worst Case unter hoher Last ausgelegt wird. Ein weiterer Nachteil von aktiver Replikation in der üblichen Form von klassischer "State Machine Replication" ist, dass alle Anfragen strikt sequentiell ausgeführt werden, was zu einer schlechten Systemauslastung führt, insbesondere bei den heutzutage weit verbreiteten Mehrkern-Architekturen.

In diesem Beitrag stellen wir unsere laufenden Arbeiten zu dynamischer vertikaler Skalierung von aktiv replizierten Diensten dar. Das Ziel dabei ist es, bei maximal möglichem Durchsatz des replizierten Systems unter Hochlast die Kosten für den Betrieb zu minimieren, insbesondere zu Zeiten von kleiner Last.

II. Konzept

Ein üblicher Ansatz zur Steigerung der Performance von Diensten ist horizontale Skalierung. Diese Herangehensweise ist allerdings für aktive Replikation weniger gut geeignet. Eine Erhöhung der Replikatzahl steigert nicht die Leistungsfähigkeit sondern erhöht stattdessen den Kommunikationsaufwand der Replikate, da dieser quadratisch mit der Zahl der Replikate wächst. Alternativ ließe sich eine weitere Replikatgruppe etablieren, die sich die Last mit der ersten Gruppe teilt. Dazu ist jedoch nicht nur ein Replikat sondern gleich eine Menge von zusätzlichen Replikaten zu installieren. Zusätzlich ist die Verteilung eingehender Anfragen an die Replikatgruppen erheblich komplexer als im nicht-replizierten Fall, da Clients oder Load-Balancer mit einer ganzen Servergruppe kommunizieren müssen und dabei kein Single-Point-Of-Failure entstehen darf.

Die Performance eines aktiv replizierten Dienstes wird von zwei Seiten beeinflusst: Auf der einen Seite erfordert die Ordnung der Anfragen durch ein Gruppenkommunikationssystem (Group Communication System – GCS) einen nicht zu vernachlässigenden Aufwand. Auf der anderen Seite wird die Performance durch den Aufwand zur Abarbeitung der Anfragen maßgeblich bestimmt. Letztere ist bei traditionellen Systemen zur aktiven Replikation durch die Leistung eines CPU-Kerns beschränkt, da alle Anfragen streng sequentiell ausgeführt werden.

In unserem Konzept zur maßgeschneiderten Optimierung der Performance eines aktiv replizierten Systems bei zugleich möglichst minimalen Kosten kombinieren wir daher mehrere Techniken. Zum einen ermöglichen wir die parallele Ausführung mehrerer Dienstanfragen durch einen optimierten deterministischen Scheduler. Dieser gestattet den Einsatz von Multithreading in der aktiv replizierten Dienstimplementierung unter Bewahrung der starken Konsistenzeigenschaften [3]. Zum anderen lässt sich die Ausführung eines Replikats durch vertikale Skalierung optimieren, indem die Zahl der verfügbaren CPUs der darunter liegenden virtuellen Maschine dynamisch angepasst wird.

Durch die vertikale Skalierung werden beide zuvor beschriebenen Probleme verringert. Auf der einen Seite lassen sich die zur Verfügung stehenden CPU-Ressourcen dem Bedarf des GCS anpassen. Diese erste Optimierung ist unkompliziert: Hier lassen sich schwellwertbasiert bei Unterlast bzw. Überlast die CPU-Ressourcen einfach anpassen. Auf der anderen Seite lassen sich auch für die Ausführung von Anfragen durch die Dienstimplementierung bei Bedarf Ressourcen bereitstellen. Hierbei wird die Optimierung durch die Wechselwirkung mit dem Scheduler erschwert, da dieser seine interne Strategie für die bekannte vorhandene CPU-Anzahl optimiert, und eine deterministische Vorhersage, ob zusätzliche CPUs durch die Anwendung ausgenutzt werden können, nicht unmittelbar möglich ist.

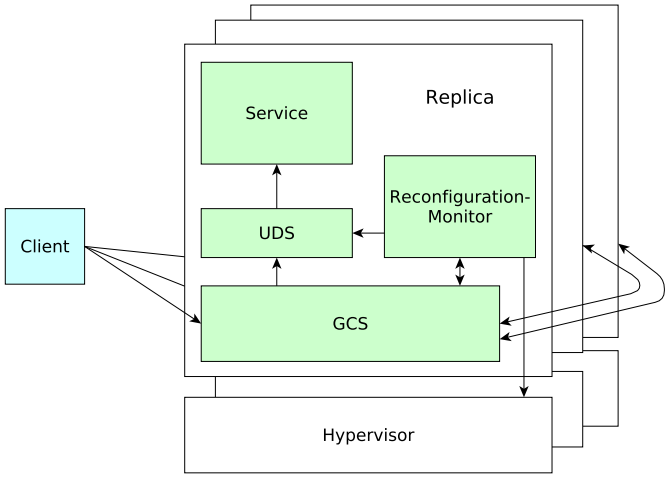

Abbildung 1. Ein Rekonfigurations-Monitor passt auf Grundlage von Monitoringdaten aus Betriebssystem und UDS-Scheduler die Zahl der virtuellen CPUs im Hypervisor dynamisch an.

III. Umsetzung

Abbildung 1 zeigt einen Überblick über eine Gesamtarchitektur, in der das beschriebene Konzept realisiert wird. Eine Umsetzung sowie experimentelle Evaluierung dieser Architektur ist derzeit in Arbeit.

Auf der Basis von BFTSMaRt [4] haben wir ein System zur aktiven Replikation entwickelt, das mit Hilfe eines UDS-Schedulers multithreaded Services ausführen kann. Dieses System wird durch einen Rekonfigurations-Monitor ergänzt, einer Entscheidungskomponente, die mit dem Hypervisor-Verwaltungssystem interagiert, um die Anzahl der virtuellen CPUs der Maschine dynamisch zu erhöhen oder zu verringern.

Die Entscheidung über eine Rekonfiguration wird unter Berücksichtigung von zwei Datenquellen getroffen. Zum einen wird auf Betriebssystemebene ermittelt, wie hoch die aktuelle CPU-Auslastung durch alle Komponenten gemeinsam (einschließlich GCS und Service) ist. Zum zweiten liefert der UDS-Scheduler Indikatoren, ob für den konkret ausgeführten Dienst eine höhere Parallelität und somit eine Verwendung weiterer CPUs möglich wäre.

Entscheidet der Monitor eine Rekonfiguration, so kann die Anpassung der virtuellen CPUs über das Hypervisor-Verwaltungssystem unmittelbar erfolgen. Gleichzeitig ist der Scheduler über die geänderte CPU-Anzahl zu informieren. Da dies das Verhalten des Schedulers beeinflusst und dieser in allen Replikaten zur Wahrung der Konsistenz identisch arbeiten muss, wird die Rekonfiguration allen Replikaten über das GCS mitgeteilt und von dort an den UDS weitergegeben.

Im Ergebnis ist es der Architektur möglich, die CPU-Anzahl periodisch bedarfsgerecht anzupassen. Damit lässt sich bei hoher Last bedarfsgerecht eine hohe Leistung erreichen. Die Kosten für Ressourcen fallen dabei gegenüber einem System, das konstant für diese Maximallast ausgelegt ist, wesentlich geringer aus.

IV. Zusammenfassung

Unsere aktuellen Forschungsarbeiten zielen darauf, die Performance von aktiver Repliation zu maximieren und zugleich die Kosten zu minimieren. Dieses Ziel erreichen wir durch eine Architektur, die dynamische vertikale Skalierung unterstützt. Eine genaue Quantifizierung der gewonnen Vorteile ist Gegenstand aktuell laufender Arbeiten.

Acknowledgment

Die vorgestellten Arbeiten wurden von der Deutschen Forschungsgemeinschaft (DFG) im Rahmen des Projekts OptSCORE gefördert.

Literatur

[1] M. Burrows, “The Chubby lock service for loosely-coupled distributed systems,” in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, 2006, pp. 335–350.

[2] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, “ZooKeeper: wait-free coordination for internet-scale systems,” in *In USENIX Annual Technical Conference*, 2010.

[3] F. J. Hauck, J. Domaschka, and G. Habiger, “UDS: a novel and flexible scheduling algorithm for deterministic multithreading,” in *Proc. of the 35th Symposium on Reliable Distributed Systems (SRDS) – accepted for publication*, 2016.

[4] A. Bessani, J. Sousa, and E. E. P. Alchieri, “State machine replication for the masses with bft-smart,” in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, June 2014, pp. 355–362.