Uma Arquitectura para Oferecer Garantias de Coerência forte a Aplicações FaaS

Rafael Soares e Luís Rodrigues

INESC-ID, Instituto Superior Técnico, U. Lisboa
{joao.rafael.pinto.soares,ler}@tecnico.ulisboa.pt

Resumo Function-as-a-Service (FaaS) é um paradigma relativamente recente, suportado por vários fornecedores de serviços na nuvem, que permite executar aplicações sem a reserva prévia de servidores. Para este efeito, as aplicações são escritas na forma de um grafo de funções efémeras, que podem ser executadas por diferentes servidores sem garantias de coerência quando acedem a serviços de armazenamento externos. Neste artigo propomos uma arquitectura para oferecer garantias transaccionais a aplicações FaaS. A arquitectura recorre a um serviço de gestão da coerência, que é usado para seriar as transacções que se executam concorrentemente. Ao contrário de trabalhos anteriores, a nossa arquitectura não obriga a que todos os acessos à memória persistente passem pelo serviço de gestão da coerência, podendo ser servidos directamente pela cama de armazenamento. Isto permite oferecer garantias de coerência com servidores de menor capacidade, preservando melhor as vantagens económicas da arquitectura FaaS. Através de uma avaliação experimental, mostramos que a nossa solução proposta oferece um débito 90% superior ao apresentado pelos protocolos alternativos.

1 Introdução

O paradigma de "computação sem servidores"(*Serverless Computing*), também conhecido como Function-as-a-Service (FaaS), é um paradigma recente, hoje oferecido por vários fornecedores de serviços da nuvem. Este paradigma permite aos programadores executar as suas aplicações na nuvem sem ter de reservar previamente servidores. Para isto, o programador deve conceber a sua aplicação na forma de um grafo de funções efémeras. Neste modelo, as funções são executadas em servidores escolhidos de forma automática pelo fornecedor do serviço, sem intervenção do cliente. Os clientes deste serviço são facturados com base no poder computacional usado, ao contrário dos modelos que obrigam os recursos a ser reservados previamente, onde o valor cobrado é proporcional ao tempo reservado, independentemente do uso real do recurso.

As arquitecturas de suporte a FaaS desagregam as camadas computacionais e de armazenamento, permitindo aos fornecedores de serviços da nuvem oferecer uma capacidade de escala elástica independente para cada camada. De modo a melhorar a capacidade de escala da camada computacional, sistemas de FaaS requerem que as funções sejam efémeras, necessitando de contactar a camada

de armazenamento para obter e partilhar estado. Este contacto constante com a camada de armazenamento leva a um custo aumentado de comunicação, levando aos fornecedores destes serviços a utilizarem serviços de armazenamento de baixa latência e alta capacidade de escala como o Anna [1] e o Redis [2]. Estes serviços oferecem coerência fraca, evitando os custo de coordenação associado com o cumprimento das semânticas de coerência forte.

Como funções diferentes podem ser executadas por nós diferentes, estes podem observar versões incoerentes dos mesmos dados obtidos do armazenamento de coerência fraca. Isto pode levar a que as aplicações observem estados intermédios e/ou incoerentes, e produzam resultados indesejados. Neste artigo abordamos o problema de estender a arquitectura FaaS com suporte que permita a execução de transacções com garantias fortes, como serializabilidade ou isolamento instantâneo. Os trabalhos anteriores que abordam este problema em ambientes semelhantes usam uma camada intermédia por onde são encaminhados todos os acessos à camada de armazenamento.

Oferecer suporte para transacções, com garantias ACID, a aplicações que usem o paradigma de FaaS pode permitir conciliar as vantagens da coerência forte com o baixo custo e a capacidade de escala que o FaaS oferece. O desafio esta em concretizar este suporte de uma forma eficiente, sem invalidar as vantagens do FaaS. Primeiro, para garantir a coerência, funções necessitam de se coordenar entre si e trocar informação sobre a captura que precisam de ler. Além disso, poderá ser necessário vários pedidos de leitura à camada de armazenamento de modo a obter uma versão coerente com as versões anteriormente lidas na mesma transacção, devido à coerência fraca da camada de armazenamento. Ao contrário desses trabalhos, a nossa arquitectura não obriga a que todos os acessos à memória persistente passem pelo serviço de gestão da coerência, podendo ser servidos directamente pela cama de armazenamento. Isto permite oferecer garantias de coerência com servidores de menor capacidade, preservando melhor as vantagens económicas da arquitectura FaaS.

2 Trabalho Relacionado

A maneira mais simples de oferecer suporte transaccional para aplicações consiste em usar um sistema de armazenamento que suporte este modelo, tipicamente uma base de dados SQL. No entanto, desenvolver um sistema de armazenamento transaccional com elevada capacidade de escala e com suporte para ajuste elástico dos recursos não é trivial. Por esta razão, muitos dos serviços de armazenamento para a nuvem não suportam transacções e, frequentemente, apenas oferecem garantias de coerência fraca, mesmo para operações individuais. Este é tipicamente o caso da maioria dos serviços de armazenamento disponíveis para quem desenvolve aplicações FaaS.

Nesta secção abordamos o estado da arte relativamente a sistemas que permitem suportar transacções sobre sistemas de armazenamento que não têm suporte nativo para esta abstracção. Cobrimos não só sistemas específicos para FaaS mas

também sistemas desenhados para outros modelos de computação na nuvem que usam armazenamento não transaccional.

2.1 Suporte Transaccional em FaaS

O desenvolvimento de suporte transaccional para FaaS é um tópico que começou recentemente a receber atenção na literatura. Nesta secção abordamos três sistemas recentes, nomeadamente, o Hydrocache [3], o AFT [4] e o Beldi[5]. Estes sistemas introduzem técnicas para superar os dois principais desafios que se levantam quando se tenta oferecer suporte transaccional em FaaS. O primeiro provém do desenho base de FaaS, em que as funções são efémeras. O segundo deriva da utilização de uma camada de armazenamento que oferece apenas coerência "alguma-vez"(do Inglês, *eventual consistency*).

O Hydrocache [3] oferece suporte para coerência transaccional causal (TCC), uma forma fraca de coerência transaccional, garantindo que todas as funções lêem versões coerentes dos objectos e aplicando as escritas, em conjunto, quando o grafo de funções termina de executar. Cada objecto é armazenado com metadados que capturam as dependências explícitas da transacção de escrita, em concreto, quais os objectos (e as versões desses objectos) que estão no passado causal da transacção de escrita. Quando uma função lê um objecto, recolhe esta informação, que é passada de função em função no grafo de funções, de forma a garantir que todas as funções lêem de um corte coerente. Para reduzir o número de acessos ao serviço de armazenamento, cada servidor mantém uma cache dos objectos lidos e escritos no passado. Este sistema, para além de não suportar transacções com coerência forte (isto é, modelos de coerência que obriguem a ordenar transacções concorrentes, como o isolamento instantâneo ou a serializabilidade), obriga à troca de muitos metadados entre as funções, o que limita o seu desempenho. Além disso, transacções podem ter de abortar por falta de objectos compatíveis com o seu conjunto de leitura, devido às leituras optimistas de cada cache.

O Atomic Fault Tolerant Shim (AFT) [4] é um sistema que ofereces garantias transaccionais para FaaS mas que opta por oferecer apenas um nível de isolamento transaccional relativamente fraco, concretamente, suporta apenas Leituras Atómicas (*Read Atomic*). Este sistema baseia-se na utilização de uma camada intermédia, que se interpõe entre as funções e o armazenamento. Esta camada é constituída por um conjunto de gestores transaccionais. Os clientes só necessitam de contactar um gestor transaccional para garantir a coerência da transacção, evitando os custos de transferência de metadados de sistemas como o Hydrocache. Cada AFT mantém um índice que mapeia cada objecto para a sua versão mais recente conhecida pelo AFT, evitando assim custos de coordenação entre AFTs em troca de leituras menos frescas. Para oferecer Leituras Atómicas, cada AFT mantém metadados para cada objecto de quais os objectos e versões que foram escritos na mesma transacção, garantindo a atomicidade das transacções e evitando leituras fracturadas. Embora a falta de coordenação entre AFTs traga benefícios em termos de latência, não só impossibilita este sistema de oferecer

garantias transaccionais fortes como poderá obrigar transacções a abortar por falta de versões compatíveis.

O Beldi [5] é um sistema que oferece suporte transaccional com garantias de opacidade[6], isto é, assegurando que as transacções nunca observam estados incoerentes, nem mesmo as transacções que abortam. Apesar dos dados serem guardados num sistema de armazenamento com garantias de coerência forte, o Beldi recorre a um sistema de de armazenamento adicional, partilhado por todas as funções, para manter metadados sobre as transacções em execução. Este serviço de armazenamento adicional, necessita de suportar a inserção atómica de múltiplos valores num histórico (*log*) de operações realizadas. Os valores registados no histórico incluem, entre outros, valores de trincos usados pelo controlo de concorrência para garantir o isolamento das transacções. As entradas neste histórico são totalmente ordenadas utilizando um protocolo semelhante a Bloqueio de 2 Fases (*2-Phase-Locking*) e esta ordem é usada para ordenar operações concorrentes que acedem aos mesmos dados. Uma desvantagem deste sistema é que todas as operações de leitura e escrita obrigam a consultar o histórico, tornando o sistema pouco eficiente.

2.2 Suporte Transaccional Para Outras Aplicações na Nuvem

O CloudTPS[7] é um sistema que oferece coerência forte por cima de um serviço de armazenamento de coerência fraca de uma forma semelhante ao AFT, recorrendo a uma camada intermédia constituída por vários "Local Transaction Managers"(LTM), em que cada LTM será responsável por uma partição do espaço de dados, fazendo a certificação de transacções que interajam com a sua partição. Os LTMs também são responsáveis por responder a pedidos de leitura, garantindo as propriedades de coerência e isolamento das transacções, dependendo do serviço de armazenamento unicamente para garantir a durabilidade das transacções. O CloudTPS utiliza um sequenciador para oferecer ordenação total das transacções, garantindo serializabilidade estrita (*Strict Serializability*). Não só o uso deste sequenciador rapidamente se torna num ponto de congestão do sistema, como a maioria das aplicações não necessita de um nível de isolamento tão forte, adicionando custos desnecessários à latência. É de notar que o CloudTPS pode ser adaptado para utilizar camadas de armazenamento de vários níveis de coerência e suporte transaccional. Para meio de comparação, consideramos que utiliza os níveis de coerência e suporte transaccional mais fracos.

A tese de doutoramento de Padhye[8] apresenta uma arquitectura semelhante à do CloudTPS para oferecer suporte transaccional forte na forma de isolamento instantâneo no ambiente de nuvem, utilizando um serviço constituído por várias réplicas de um serviço de detecção de conflitos, cada um responsável por assegurar que não existem conflitos entre transacções concorrentes de um certo espaço de objectos. Ao contrário do CloudTPS, Padhye utiliza uma camada de armazenamento de coerência forte para responder a pedidos de leitura e manter versões de objectos de transacções ainda por confirmar. Assim, cada réplica do serviço de detecção de conflitos só mantém informação sobre a versão mais recente de cada objecto, utilizando esta informação para identificar conflitos de escrita. Padhye

também utiliza um sequenciador para fornecer a estampilha temporal do corte coerente que deve ser observado por uma transacção. No entanto, este sequenciador mantém informação do estado de todas as transacções, fornecendo sempre estampilhas referentes a cortes estáveis, i.e, cortes em que todas as transacções no passado já tenham terminado. Embora o serviço de detecção de conflitos de Padhye utilize menos memória que outras alternativas (pois não mantêm os dados dos objectos), o uso da camada de armazenamento forte introduz latência extra a cada pedido de leitura. Além disso, o uso do sequenciador mantém-se como um ponto de congestão do sistema.

2.3 Estampilha Temporal de Confirmação

Os sistemas anteriores utilizam um sequenciador de modo a obter uma estampilha temporal de confirmação. Este método não só introduz custos de latência na forma de rondas de comunicação extra como também introduz um ponto de congestão do sistema. Num ambiente de alta capacidade de escala e em que baixa latência é um requisito, pretendemos minimizar estes custos ao máximo.

O ClockSI[9] é um protocolo que oferece isolamento instantâneo a sistemas particionados recorrendo a relógios físicos sincronizados, permitindo às varias partições fornecer a clientes uma estampilha temporal do corte coerente de leitura sem necessitar de executar uma ronda de coordenação explícita entre as partições. Além disso, este protocolo não necessita de um sequenciador, sendo os marcos temporais de confirmação das transacções negociado entre as partições através de num protocolo de confirmação em duas fases. Devido aos potenciais desvios dos relógio entre partições distintas, os pedidos de leitura poderão usar uma estampilha temporal que está no futuro de uma dada partição, obrigando o pedido de leitura a bloquear até que o relógio da partição alcance o valor pré-definido para a leitura. Utilizando técnicas de sincronização de relógios comuns como NTP[10], o tempo máximo de bloqueio poderá ser na ordem dos milissegundos, introduzindo um custo alto de latência adicional em FaaS.

A Tabela 1 apresenta um sumário do trabalho relacionado. Como podemos ver, existem vários sistemas que oferecem suporte transaccional em vários ambientes e condições. No entanto, em FaaS nenhum sistema conseguiu oferecer isolamento forte de uma forma eficiente (Beldi oferece latências na ordem dos segundos enquanto outros sistemas estão nos milisegundos). Além disso, embora sistemas da nuvem mostram exemplos de suportes transaccionais por cima de vários níveis de camadas de armazenamento na forma de uma camada intermédia, o uso excessivo desta camada leva a um custo extra na falta de capacidade de escala, algo cujo peso é maior num ambiente como FaaS.

3 FaaSSI

Nesta secção apresentamos o FaaSSI, um sistema para fornecer suporte transaccional, com garantias de coerência forte em ambientes FaaS. Tal como a maioria dos trabalhos anteriores[4,7,8], o nosso sistema recorre a uma camada

	Ambiente	Armazenamento Coerência Forte Transaccional	Nivel Isolamento	Pedidos de Leitura	Rondas de Leitura
Beldi	FaaS	✓	Opacidade	Armazenamento	1
Padhye	Nuvem	✓	Isolamento Instantâneo	Armazenamento	1
HydroCache	FaaS	✗	TCC	Armazenamento	≥1
AFT	FaaS	✗	Leituras Atómicas	Gestor transaccional	1
CloudTPS	Nuvem	✗	Serializabilidade Estrita	Gestor transaccional	1
FaaSSI	FaaS	✗	Isolamento Instantâneo	Gestor transaccional + Armazenamento	≈1

Tabela 1: Comparação do estado da arte

intermédia, que se posiciona entre os nós que executam as transacções e o sistema de armazenamento. Esta camada realiza, entre outras funcionalidades, o controlo de concorrência. A arquitectura pode ser configurada para oferecer diferentes variantes de coerência forte mas, neste trabalho, focamos-nos em oferecer isolamento instantâneo.

O FaaSSI foi desenhado para se executar em centros de dados, tendo sido desenvolvido como uma extensão ao sistema Cloudburst [11]. Uma maneira de oferecer isolamento instantâneo a aplicações que executam no Cloudburst seria recorrer a uma camada de armazenamento transaccional (por exemplo, uma base de dados tradicional). Isto seria demasiado pesado para as aplicações que não necessitam de garantias tão fortes. Por esta razão, para serviço de armazenamento escolhemos o Anna [1] que só oferece garantias de coerência "alguma-vez". Esta escolha tem a vantagem de permitir ao utilizador, num único sistema, poder escolher o modelo de coerência mais adequado à sua aplicação. Caso pretenda coerência fraca pode aceder ao Anna sem recorrer a nenhum serviço adicional e sem sofrer nenhuma penalização no desempenho, se pretender ter coerência transaccional causal pode usar um sistema como o Hydrocache [3], e se pretender isolamento instantâneo usa o FaaSSI. Ilustramos o nosso sistema na Figura 1.

3.1 Arquitectura

A camada intermédia é constituída por um conjunto de Gestores de Conflitos, responsáveis por verificar em tempo de confirmação se uma transacção satisfaz o isolamento instantâneo. Nós seguimos uma abordagem semelhante ao CloudTPS [7], onde cada gestor de conflitos é responsável por uma partição do espaço de dados. Os gestores de conflitos são também responsáveis por tornarem as escritas persistente no Anna: estes fixam-se a uma replica do Anna para cada objecto, podendo assim sempre recuperar o valor mais recente de um objecto.

Finalmente, os gestores de conflitos mantêm uma *cache* de versões e uma cache de conteúdo dos últimos objectos escritos para ajudarem os clientes a ob-

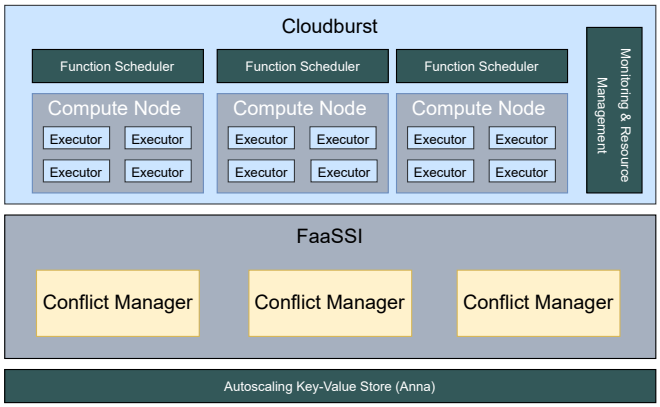


Figura 1: Camada transaccional FaaSSI. Baseado no diagrama de [3]. Secções amarelas representam as adições do FaaSSI.

terem cópias actualizadas dos objectos mais populares. O tamanho das caches dos gestores de conflitos é um parâmetro do sistema. Quando a memória se esgota, são descartados objectos de acordo com uma política de substituição que pode também ser configurada. Na actual versão do sistema usamos exclusivamente a política LRU (*Least Recently Used*) como estratégia de substituição.

O FaaSSI concretiza um protocolo de confirmação atómica semelhante ao usado no ClockSI [9], que recorre a relógios sincronizados para ordenar as transacções concorrentes. Este tipo de protocolo pode introduzir atrasos a pedidos de leitura que são proporcionais à precisão do algoritmo de sincronização de relógios. A evolução dos algoritmos de sincronização de relógios para centros de dados, da qual o sistema Sundial [12] é um bom exemplo, permite hoje sincronizar relógios com uma precisão na ordem dos $100ns$, tornando o protocolo bastante eficiente.

3.2 Operações de Leitura

Uma transacção lê o estado coerente do sistema de armazenamento num instante temporal que é definido quando a transacção tem início. Este instante é obtido calculando o máximo entre a estampilha associada à última transacção de escrita do cliente e o valor da leitura do relógio sincronizado realizado pelo agendador de funções do Cloudburst antes de iniciar o grafo de funções. Isto assegura que o cliente vê sempre as suas próprias escritas.

No FaaSSI, quando um nó de processamento necessita de ler um objecto, faz um pedido de leitura ao Anna e, em paralelo, contacta o gestor de conflitos correspondente para obter a versão mais recente do mesmo. Isto é necessário pois o Anna apenas dá garantias de coerência "alguma-vez"e pode não retornar a versão mais recente do objecto. Se a versão retornada pelo Anna for coerente com a versão indicada pelo gestor de conflitos, a leitura termina. Caso contrário, o nó de processamento, volta a contactar o gestor para obter o conteúdo da versão

mais actual (como referimos anteriormente, o gestor de conflitos consegue sempre extrair esta versão, mesmo que não esteja em cache).

Nos casos em que a versão retornada pelo Anna é incoerente, o método adoptado pelo FaaSSI introduz uma latência adicional em relação a uma versão em que todas as leituras são feitas sempre através da camada intermédia. No entanto, como veremos, para a maioria dos padrões de utilização, este não é o caso mais comum. Pelo contrário, o Anna retorna frequentemente versões coerentes. Desta forma, o método adoptado pelo FaaSSI permite reduzir de forma significativa a carga na camada intermédia e, inclusive, reduzir a latência: nos casos em que o objecto não está na cache da camada intermédia, é mais eficiente ler directamente do Anna.

Tal como no Clock-SI, caso uma transacção concorrente esteja a confirmar um valor que possa pertencer ao corte coerente pretendido, o gestor de conflitos terá de esperar que essa transacção termine antes de poder retornar as versões dos objectos correctos ao cliente. Caso a transacção concorrente confirme com sucesso, o gestor retorna ao cliente os valores dos objectos em vez de apenas retornar as respectivas versões. Esta optimização justifica-se pelo facto de neste cenário, em que a transacção que gerou os valores a ler acabou de confirmar, a probabilidade de o Anna devolver ao cliente as versões mais recentes ser baixa.

3.3 Operações de Escrita

Quando um grafo de funções se executa com requisitos de isolamento instantâneo, todas as escritas são armazenadas num *buffer*, que é passado de função em função. A função de escoamento do grafo (isto é, a última função do grafo) envia os dados escritos pela transacção, conjuntamente com a versão do sistema utilizada para realizar as leituras, para os gestores de conflitos correspondentes. Se o espaço de dados do conjunto de escrita pertencer a vários gestores de conflitos, um deles é eleito, de forma determinista, para coordenar a confirmação, recorrendo a um protocolo de confirmação em duas fases. Caso a transacção seja confirmada, os gestores de conflitos escrevem as actualizações para a camada de armazenamento.

3.4 Protocolo de Confirmação para Isolamento Instantâneo

Quando a transacção termina, cada gestor de conflitos atribui uma estampilha temporal provisória à transacção, correspondente ao valor do relógio sincronizado. Esta estampilha é usada para verificar se existe um conflito com algum dos objectos mantidos por esse gestor de conflitos. Se a transacção passar a certificação local, a estampilha provisória é enviada para o coordenador.

A estampilha final, usada para seriar as transacções concorrentes, é calculada pelo gestor coordenador usando o máximo de todas as estampilhas provisórias. Este processo permite confirmar se as versões lidas são ainda válidas no instante correspondente à estampilha final. Em caso afirmativo, a transacção confirma e cada gestor de conflitos persiste o estado dos objectos no Anna. Caso contrário, as versão geradas são descartadas e a transacção tem de se reiniciar.

Durante a execução do protocolo de confirmação em duas fases, os objectos escritos pela transacção são bloqueados, e outras transacções que tentem escrever sobre os mesmos dados têm de esperar. Isto é necessário para assegurar que no caso de duas ou mais transacções escreverem de forma concorrente num dado objecto, apenas uma delas irá confirmar. Para evitar o interbloqueio de transacções que tentam escrever sobre os mesmos dados concorrentemente, usa-se uma estratégia de "espera-ou-morte"(do Inglês, *wait-die*) em que a transacção com a estampilha mais recente aborta e as transacções com estampilhas anteriores esperam.

3.5 Tolerância a Faltas

Várias técnicas de tolerância a faltas a este tipo de sistemas foram abordados no estado da arte, como em [7,8]. As técnicas apresentadas neste artigo são ortogonais à estratégia de tolerância a faltas. Por brevidade, não abordamos este tema neste artigo.

4 Avaliação

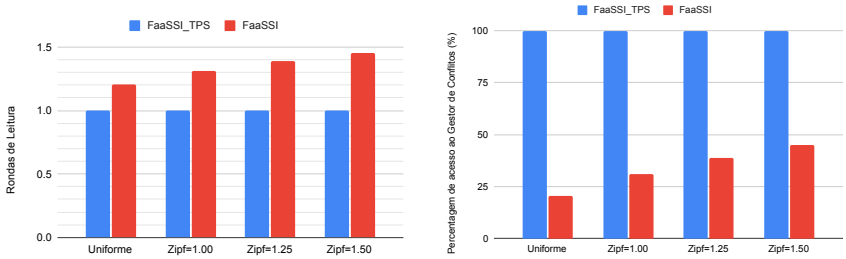
Nesta secção iremos avaliar o desempenho do FaaSSI. Iremos comparar várias versões do FaaSSI, executando algoritmos diferentes, de modo a aferir as potencias vantagens e limitações da técnica aqui proposta.

4.1 Bancada Experimental e Padrões de Carga

A nossa avaliação é baseada em execuções de um protótipo do FaaSSI na plataforma experimental Grid5000[13] usando os seus servidores dedicados. Cada servidor é constituído por 1 Intel Xeon Gold 5220 CPU com 18 núcleos, 96GB RAM e 480GB de armazenamento SSD. Os servidores estão ligados por comutadores de $25Gbps$. A latência observada entre grupos de servidores foi de aproximadamente $0.15\ ms$. Os relógios foram sincronizados utilizando o NTP.

Para estas experiências conseguimos reservar 17 máquinas físicas que foram usadas para correr múltiplas máquinas virtuais, atribuídas da seguinte forma: 3 para executar clientes, 4 para executar as funções do Cloudburst, 8 para o sistema de armazenamento Anna, 1 para executar o nosso gestor de conflitos, e 1 onde executam os gestores do sistemas (tais como o escalonador e a monitorização do Cloudburst, entre outros).

A carga experimental utilizada foi baseada nos testes usados em [3]. As experiências foram executadas com 12 clientes concorrentes. Cada cliente executa 2000 pedidos de grafos de funções sequencialmente, constituídos por 3 funções, cada uma executando 4 leituras, dando um total de 12 leituras por transacção. Para alem destes pedidos, cada cliente tem uma probabilidade de executar um pedido de grafo de escrita, escrevendo em 10 objectos. Utilizamos um rácio de leitura/escrita de 33%, valor semelhante ao utilizado por cargas-padrão como TPC-C. O conjunto de dados é constituído por 10000 chaves, cada uma com o



(a) Média rondas de pedido de leitura

(b) Percentagem de acessos ao Gestor de Conflitos

Figura 2: Acessos de leituras

conteúdo de 2048 octetos. O conjunto de dados foi dividido por 32 partições, com um factor de replicação de 3 (i.e. cada chave tem 3 replicas) e um tempo de propagação de actualizações de 3 segundos (i.e. cada partição propaga as actualizações das suas réplicas a cada 3 segundos). Lançamos 12 nós executores, cada um com 4 fios de execução (tal como ilustrado na Figura 1), de modo a que cada cliente tenha sempre um executor disponível para executar o seu grafo. Removemos a cache de dados do Cloudburst de modo a não influenciar o sistema. Limitamos a largura de banda do Gestor de Conflitos para $100mbit/s$ de modo a mais facilmente congestionar o serviço, dado que seria necessário um número de máquinas cliente muito superior ao que temos disponível para congestionar naturalmente o sistema.

4.2 Débito das transacções de Leitura

Começamos por avaliar a nossa proposta em comparação com os sistemas anteriores. Mais especificamente, comparamos com técnicas que colocam toda a carga de leitura na camada intermédia como no CloudTPS[7] e outros[4]. Para tal, desenvolvemos uma variante do FaaSSI, a que chamamos FaaSSI-TPS, que concretiza estes algoritmos. Para ambos os sistemas, avaliamos o débito de transacções de leitura com um único Gestor de Conflitos e desactivamos a cache de valores, dando memoria máxima à cache de versões. Testamos com um único Gestor de Conflitos para isolar o efeito do protocolo de leitura proposto de outros factores ortogonais à nossa contribuição. Variamos a probabilidade de acesso as chaves seguindo uma distribuição Zipfiana e uma distribuição uniforme. Cada cliente apresenta esta distribuição em conjuntos de objectos diferentes. Nenhuma transacção requisita o mesmo objecto mais que uma vez de modo a garantir um maior leque de acesso aos objectos.

Comecemos por analisar o numero de rondas médio executadas, como observado na Figura 2a. Como podemos observar, o número de rondas de leitura varia entre 1.20 e 1.45 rondas de leitura média, com um aumento do número de

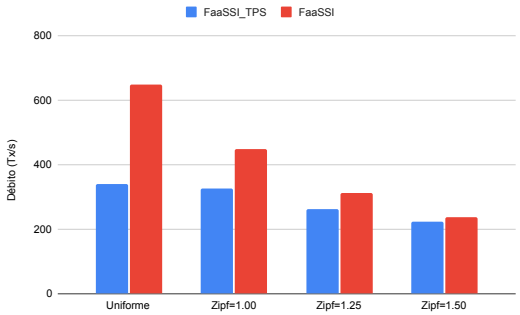


Figura 3: Débito do FaaSSI vs FaaSSI-TPS

rondas com o aumento da zipfiana. Com o maior enviesamento de acesso aos dados, maior será a probabilidade de uma transacção ler um objecto recentemente escrito. Por consequência, como podemos observar na Figura 2b, o FaaSSI contacta o gestor de conflitos para obter o valor de um objecto em apenas 20% dos pedidos, diminuindo assim significativamente o número de vezes em que o gestor de conflitos tem de fornecer os dados ao cliente comparativamente ao FaaSSI-TPS.

A Figura 3 mostra o débito do sistema para as várias configurações. Como podemos observar, quando utilizando uma distribuição uniforme de acessos, o FaaSSI apresenta um débito 90% mais alto que o FaaSSI-TPS. Isto mostra que a nossa proposta, ao reduzir a carga no gestor de conflitos, leva a um melhor desempenho por parte do FaaSSI.

Conseguimos ver uma diminuição do débito com o aumento do enviesamento no acesso a dados. Esta diminuição deve-se a dois factores. No caso do FaaSSI, o aumento do enviesamento leva a que sejam necessárias mais rondas de leitura, pois existe uma maior probabilidade de acedermos a objectos que tenham sido recentemente actualizados. Para além disso, em ambos os casos, o aumento o enviesamento leva a um maior peso nas replicas do Anna, que terão mais acessos por parte do gestor de conflitos que, estando fixado a réplicas, levara a uma maior carga e como tal pior desempenho.

5 Conclusão

A computação sem servidores é um novo paradigma para desenvolvimento de aplicações, oferecendo uma capacidade de escala mais eficiente e com maior granularidade sem as preocupações de alocação de infraestruturas. Os sistemas actuais não oferecem suporte para aplicações que querem utilizar este paradigma mas necessitam de garantias fortes e alto desempenho. Neste artigo apresentamos o FaaSSI, um sistema que oferece garantias transaccionais fortes no ambiente de FaaS recorrendo a um serviço de gestão de coerência. Desenvolvemos um novo protocolo de leitura que reduz a carga do serviço de gestão de coerência, oferecendo uma melhor capacidade de escala e custos reduzidos comparativamente

aos sistemas anteriores. A nossa solução consegue reduzir a carga do gestor de conflitos no suporte a pedidos de leitura. Mostrámos que, para padrões de acesso pouco enviesados, isto permite obter um aumento no débito de cerca de 90%. A possibilidade de conseguir maior débito também com cargas mais enviesadas está ainda em estudo. No futuro, pretendemos também avaliar o FaaSSI com vários gestores de conflitos.

Agradecimentos: Agradecemos a Taras Lykhenko pelos vários comentários e sugestões durante o desenvolvimento deste trabalho. Este trabalho foi suportado pela FCT – Fundação para a Ciência e a Tecnologia,através dos projectos UID/ CEC/ 50021/ 2020 e COSMOS (financiado pelo OE com a ref. PTDC/ EEICOM/ 29271/ 2017 e pelo Programa Operacional Regional de Lisboa na sua componente FEDER com a ref. Lisboa-01-0145-FEDER-029271).

Referências

1. Wu, C., Faleiro, J., Lin, Y., Hellerstein, J.: Anna: A KVS for any scale. In: ICDE, Paris, France (April 2018) 401–412
2. Redis. https://redis.io/ Accessed: 11/12/2020.
3. Wu, C., Sreekanti, V., Hellerstein, J.M.: Transactional causal consistency for serverless computing. In: SIGMOD, Portland (OR), USA (June 2020) 83–97
4. Sreekanti, V., Wu, C., Chhatrapati, S., Gonzalez, J.E., Hellerstein, J.M., Faleiro, J.M.: A fault-tolerance shim for serverless computing. In: EuroSys, Heraklion, Greece (April 2020)
5. Zhang, H., Cardoza, A., Chen, P.B., Angel, S., Liu, V.: Fault-tolerant and transactional stateful serverless workflows. In: OSDI, USENIX Association (November 2020)
6. Guerraoui, R., Kapalka, M.: On the correctness of transactional memory. In: PPoPP, Salt Lake City (UT), USA (2008) 175–184
7. Zhou, W., Pierre, G., Chi, C.H.: Cloudtps: Scalable transactions for web applications in the cloud. IEEE Trans. Serv. Comput. **5**(4) (January 2012) 525–539
8. Padhye, V.: Transaction and data consistency models for cloud applications. Ph.D., University of Minnesota, Minneapolis, MN, USA (February 2014)
9. Du, J., Elnikety, S., Zwaenepoel, W.: Clock-SI: Snapshot isolation for partitioned data stores using loosely synchronized clocks. In: SRDS, Braga, Portugal (October 2013) 173–184
10. The network time procotol. http://www.ntp.org/ Accessed: 07/07/2021.
11. Sreekanti, V., Wu, C., Lin, X.C., Schleier-Smith, J., Gonzalez, J.E., Hellerstein, J.M., Tumanov, A.: Cloudburst: Stateful functions-as-a-service. Proc. VLDB Endow. **13**(12) (July 2020) 2438–2452
12. Li, Y., Kumar, G., Hariharan, H., Wassel, H., Hochschild, P., Platt, D., Sabato, S., Yu, M., Dukkipati, N., Chandra, P., Vahdat, A.: Sundial: Fault-tolerant clock synchronization for datacenters. In: OSDI. (November 2020) 1171–1186
13. Balouek, D., Amarie, A.C., Charrier, G., Desprez, F., Jeannot, E., Jeanvoine, E., Lèbre, A., Margery, D., Niclausse, N., Nussbaum, L., Richard, O., Perez, C., Quesnel, F., Rohr, C., Sarzyniec, L.: Adding virtualization capabilities to the Grid'5000 testbed. In: CLOSER, Porto, Portugal, Springer (April 2012)