Rumo à gestão de dados de observabilidade em escala

Suman Karumuri
Slack Technologies
skarumuri@slack-corp.com

Franco Solleza, Universidade Stan
Zdonik Brown
{fsolleza,sbz}@cs.brown.edu

Nesime Tatbul
Laboratórios Intel e MIT
tatbul@csail.mit.edu

RESUMO

A observabilidade vem ganhando importância como uma capacidade essencial nos sistemas e serviços de software de larga escala da atualidade. Motivado pela experiência atual da indústria, exemplificada pelo Slack, e como um chamado à pesquisa em banco de dados, este artigo descreve os desafios e oportunidades envolvidos no projeto e na construção de Sistemas de Gerenciamento de Dados de Observabilidade (SGDOs) para lidar com essa carga de trabalho emergente em escala.

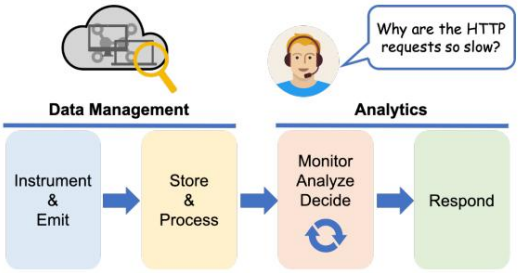


Figura 1: Observabilidade

1 INTRODUÇÃO Em 12 de maio de 2020, às 16h45 PST, a plataforma de comunicação empresarial baseada em nuvem Slack sofreu uma interrupção total de serviço [9]. Para seus milhões de usuários, a interrupção durou 48 minutos; dentro do Slack, a cascata de eventos que levaram a essa interrupção começou às 8h45 PST. Tudo começou com um bug de desempenho de software que foi detectado e imediatamente revertido durante uma implantação de código de rotina. Isso acionou o dimensionamento automático da camada da web do Slack, aumentando-o para mais instâncias do que o limite rígido permitido pelo balanceador de carga. Isso, por sua vez, expôs um bug em como o Slack atualiza a lista de hosts no balanceador de carga: alguns balanceadores de carga tinham uma mistura de instâncias de host antigas, obsoletas e novas não registradas. Oito horas depois, as únicas instâncias de host ativas eram as mais antigas ainda registradas nos balanceadores de carga. Quando o programa de escalonamento automático começou a reduzir o escalonamento dos hosts durante a noite, ele desligou essas instâncias antigas. Como todas as instâncias restantes registradas nos balanceadores de carga estavam obsoletas ou eram novas e não registradas, o serviço sofreu uma interrupção total.

Embora esta descrição do incidente do Slack apresente a sequência lógica de eventos que levaram à interrupção, identificar a causa raiz do problema exigiu “todos os esforços” [9]. Assim que o alerta foi emitido, engenheiros de diversas equipes se reuniram e exploraram diversas hipóteses possíveis com base em dados operacionais visíveis por meio de sua infraestrutura de monitoramento, painéis e alertas. Este incidente ilustra a Observabilidade em ação, uma capacidade crítica não apenas na Slack, mas em muitas outras grandes empresas de software [2].

Os atuais sistemas e serviços de software voltados para o usuário e em escala web (por exemplo, Slack, Google, Facebook, Twitter) são construídos e operados em microsserviços gerenciados por infraestruturas altamente elásticas e compartilhadas. Esse ecossistema de software nativo da nuvem está cada vez mais distribuído, heterogêneo e complexo, tornando desafiador prever seu comportamento diante de falhas e cargas variáveis [1]. A observabilidade está emergindo. como uma capacidade essencial para monitorar e manter sistemas nativos da nuvem para garantir sua qualidade de serviço aos clientes [25]. Emprestada da teoria do controle, a noção de observabilidade traz melhor visibilidade para a compreensão do comportamento complexo do software usando telemetria coletada do sistema em tempo de execução [14]. Além do simples monitoramento de caixa-preta, a observabilidade fornece insights contextuais mais profundos sobre a correção e o desempenho dos sistemas. Seu objetivo é minimizar o tempo para insight - uma medida crítica para entender o que está acontecendo no sistema e por quê. Como tal, a observabilidade é inerentemente um processo intensivo em dados e sensível ao tempo que envolve humanos no circuito (por exemplo, equipes de DevOps) [27].

Vemos a observabilidade como um problema de gerenciamento de dados. Os sistemas são instrumentados para gerar grandes volumes de dados de séries temporais heterogêneas que devem ser indexados, armazenados e consultados em tempo quase real. A instrumentação pode emitir quatro tipos de dados: (i) dados numéricos, como medidores e contagens; (ii) dados altamente estruturados sobre eventos do sistema; (iii) logs de strings não estruturadas; e (iv) um gráfico do caminho de execução de uma solicitação. Na indústria, esses dados são chamados de Métricas, Eventos1, Logs e Rastreamentos.

Como a observabilidade é crucial para o cumprimento dos objetivos de nível de serviço (SLOs) para empresas da web com milhões de usuários, há muita atividade industrial nesse domínio. No entanto, as soluções atuais consistem em uma mistura de diversas ferramentas especializadas para atender às diferentes necessidades de cada tipo de dado de série temporal. Essas soluções ad hoc não escalam bem e incorrem em altos custos de desempenho, complexidades operacionais e infraestrutura [13]. Há uma necessidade crescente de repensar o design atual das infraestruturas de dados e software para permitir o gerenciamento de dados de observabilidade em escala.

Neste artigo de visão, analisamos os requisitos de gerenciamento de dados de cargas de trabalho de observabilidade (§2) e os desafios

1Muitos profissionais de observabilidade representam eventos como sequências altamente estruturadas e um subconjunto de logs. No §2, descrevemos como a natureza estruturada dos eventos justifica uma consideração separada dos logs.

Dados	Tipo	Consultas	Armazenar	Volume	Retenção
Métricas	numérico metadados de string	agregações filtros em metadados	séries temporais comprimidas armazenamento de coluna híbrida	4B séries temporais/dia 12 milhões de amostras/segundo 12 TB/dia (compactado)	30 dias
Eventos de strings altamente estruturadas ou filtros binários em		correspondências exatas de strings armazenadas em colunas		250 TB/dia (bruto)	3-24 meses
Registra strings semiestruturadas aproximando a busca de strings			índice invertido	90 TB/dia (bruto)	7 dias
Rastreia DAGs de durações de execução de pesquisa		de gráfico dissociado	colunar / índice invertido 2 TB/dia (bruto)		14 dias

Tabela 1: As quatro categorias de séries temporais de observabilidade (MELT) diferem amplamente em suas características e necessidades

experimentado pelos sistemas atuais usando a infraestrutura de observabilidade do Slack como exemplo (§3). Em seguida, identificamos o princípios de design para a construção de ODMSs em escala web e fornecer o projeto de uma nova arquitetura para realizá-los (§4).

2 Métricas, Eventos, Registros, Rastreamentos

O desafio fundamental dos ODMS é a compreensão oportuna de o estado de um sistema diante de grandes volumes de dados e heterogeneidade. Esta seção analisa dados de observabilidade em quatro categorias: Métricas, Eventos, Logs e Rastros (MELT). Nós primeiro discuta as características únicas dessas categorias, seguidas de seus requisitos comuns para gerenciamento de dados.

2.1 Métricas

metrics	tags	timestamp	value
http	dc="dc1" host="h1" path="/"	1574260177	124
http	dc="dc1" host="h2" path="/"	1574260170	109
http	dc="dc1" host="h1" path="/"	1574260215	116
http	dc="dc2" host="h1" path="/"	1574260236	105

Figura 2: Uma métrica para o número de solicitações HTTP/segundo

As métricas fornecem medições quantitativas do desempenho e da disponibilidade do sistema em um ponto específico no tempo. Elas abrangem três tipos de dados numéricos: (i) contadores são valores que só pode aumentar ou ser zerado (por exemplo, número total de solicitações HTTP recebidas); (ii) os medidores são valores que podem ir para cima ou para baixo para refletir o estado do sistema (por exemplo, o número de HTTP solicitações aguardando resposta); e (iii) histogramas observações de amostra em um intervalo de tempo fixo, contando-as em intervalos configuráveis (por exemplo, durações de solicitações HTTP ou tamanhos de resposta). Além de um valor numérico e um carimbo de data/hora, as métricas também incluem metadados como um conjunto de pares de chave-valor, chamadas tags. As tags identificam uma instância específica de uma métrica. Uma combinação única de uma métrica e tags é chamada de série temporal. Um par de timestamp e valor é chamado de amostra. Figura 2 fornece um exemplo de métrica, http, medindo o número de Solicitações HTTP por segundo. Cada medição é marcada com os metadados do centro de dados, do host e do caminho para a origem da solicitação . Há três séries temporais (codificadas por cores) neste exemplo, cada um com carimbos de data/hora aumentando monotonicamente.

As métricas são usadas de duas maneiras: (i) para gerar alertas sobre estados inesperados do sistema ou (ii) para fins analíticos e de painel. consultas. Os alertas são gerados usando pequenas consultas sobre os mais dados atuais (por exemplo, número total de solicitações HTTP por host por minuto). Consultas analíticas podem envolver dados de fontes arbitrárias vezes para observar tendências em todo o sistema (por exemplo, número total de Solicitações HTTP por host por minuto em dc1 no último dia). Análises mais complexas (por exemplo, pesquisa de similaridade ou agrupamento [18, 21]) também pode ser realizada.

As métricas exigem um mecanismo de armazenamento híbrido. Prometheus [20], um mecanismo de armazenamento métrico amplamente utilizado, emprega um compactado estratégia de armazenamento para os valores da métrica e um índice invertido para as tags associadas. Essa estratégia resulta em altas taxas de compressão e operações de filtragem rápidas. Outras estratégias de armazenamento (por exemplo, armazenamento de séries de dados [17]) também foram propostos. Slack gera cerca de 4 bilhões de séries temporais por dia, a uma taxa de 12 milhões de amostras por segundo, coletando 12 TB (compactados) de Dados de métricas todos os dias. Eles são armazenados por 30 dias.

2.2 Eventos

Eventos são dados altamente estruturados emitidos durante o tempo de execução. Frequentemente, eles vêm de um conjunto finito de valores possíveis. Por exemplo, existem 9 métodos de solicitação HTTP (por exemplo, GET, POST) e um conjunto finito de códigos de status de resposta (por exemplo, “404: Não encontrado”). Eventos também podem ser usados para eventos de alta cardinalidade dados com dimensões superiores (por exemplo, IDs de clientes e rede endereços). Como os eventos são emitidos como strings estruturadas ou em um formato binário compacto [25], a literatura anterior sobre observabilidade normalmente considera os eventos como uma subcategoria de logs (discutido em §2.3) [12, 25]. Nós os categorizamos separadamente, porque o modelo de dados, consultas e padrões de acesso para eventos são substancialmente diferentes daqueles para logs.

A Figura 3 mostra um trecho de eventos brutos emitidos por um sistema que manipula solicitações e respostas HTTP.

```
20 de novembro de 2019, 17:35:23: 192.168.100.11 CAMINHO DA POSTAGEM:/ 200 OK
20 de novembro de 2019, 17:35:24: 192.168.100.10 OBTER CAMINHO:/ 200 OK
20 de novembro de 2019, 17:35:27: 192.168.100.10 DC=2 HOST=2 ERRO 505
20 de novembro de 2019, 17:35:28: 192.168.101.11 CAMINHO DA POSTAGEM:/ 200 OK
20 de novembro de 2019, 17:35:28: 192.168.101.10 OBTER CAMINHO:/ui/ 404
```

Figura 3: Eventos emitidos por um servidor HTTP

Além das tendências de computação, os eventos são normalmente usados para identificar instâncias específicas de estado inesperado do sistema. Por exemplo , uma consulta SQL como SELECT * from Events WHERE ip=192.168.100.11 e o método="POST" mostraria todas as solicitações POST do endereço IP consultado 192.168.100. 11.

A natureza estruturada dos dados e as consultas de filtro de baixa seletividade tornam os armazenamentos de colunas ideais para eventos. Os dados em um armazenamento de eventos geralmente são acessados por meio de consultas SQL em um armazenamento de colunas. O Slack gera mais de 250 TB de dados brutos de eventos por dia e armazena mais de 70 PB de dados simultaneamente. Os dados de eventos são retidos por períodos relativamente mais longos (3 a 24 meses) para fins de arquivamento ou auditoria jurídica.

2.3 Logs

Logs são coleções de strings semiestruturadas ou não estruturadas. Eles expõem informações altamente granulares com rico contexto local . Essa flexibilidade torna os logs cruciais para entender por que um comportamento inesperado ocorreu em um serviço. Por exemplo, ao responder ao incidente de interrupção, o Slack se baseou em logs para identificar o bug na atualização da lista de hosts do seu balanceador de carga.

Da mesma forma, para uma solicitação HTTP, um desenvolvedor de aplicativo pode incluir um rastreamento de pilha junto com uma exceção mostrando o estado do aplicativo em um log de erros (veja a Figura 4).

```
Qua, 20 de nov de 2019, 17:35:22: GET / ERRO 500
InternalServerErrorException: HTTP 500 ... em
    ServiceUnavailableException ... em RedirectionException ...
```

Figura 4: Entrada de log com rastreamento de pilha

Ao contrário das métricas, os logs geralmente contêm informações contextuais que podem fornecer respostas mais detalhadas a perguntas como: "Qual erro do servidor fez com que a resposta tivesse o status 500?". Essas consultas do tipo "agulha no palheiro" são fundamentalmente diferentes das consultas de correspondência exata feitas sobre eventos.

Eles são mais bem atendidos usando soluções de armazenamento baseadas em índice invertido devido à necessidade de correspondências aproximadas de strings. O Slack coleta cerca de 90 TB de dados de log/dia para serem retidos por 7 dias.

2.4 Traços

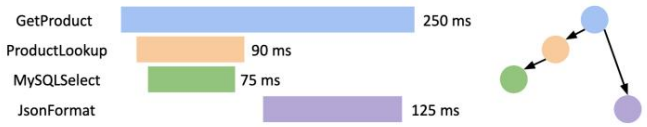
Rastros encapsulam informações sobre o caminho de execução de uma solicitação, de forma semelhante aos gráficos de chamadas [8, 11, 23, 24]. Em microsserviços e ambientes distribuídos, os rastros são gráficos de chamadas entre serviços distribuídos e incluem invocações RPC, filas assíncronas e outras comunicações entre serviços. Rastros são representados como gráficos acíclicos direcionados (DAGs), onde um vértice representa uma unidade de execução (por exemplo, uma chamada de função) chamada de intervalo e as arestas indicam uma ordenação causal (ou seja, o " acontece antes" de Lamport [10]) de um vértice para outro. Essa definição é cada vez mais aceita com esforços em toda a indústria, como o OpenTelemetry [14].

A Figura 5 mostra um trecho de um rastreamento de uma solicitação de pesquisa de produto por meio do serviço GetProduct. Para responder a essa solicitação, o GetProduct chama o serviço ProductLookup, que, por sua vez, realiza uma chamada ao banco de dados MySQLSelect. O GetProduct então formata o resultado e, por fim, responde à solicitação.

O trace e seus spans encapsulam a estrutura do caminho de execução desta solicitação e informações sobre esse caminho. Os spans capturam a duração da execução e os metadados armazenados como tags na forma de pares chave-valor.

Trace ID	Span ID	Parent ID	Start Time (ms)	Duration (ms)	Name	Tags
1234	0		1574260177000	250	GetProduct	api=/getproduct, product=7
1234	1	0	1574260177020	90	ProductLookup	product=7, query=lookup
1234	2	1	1574260177030	75	MySQLSelect	query=select
1234	3	0	1574260177110	125	JsonFormat	encoding=json

(a) Uma tabela de intervalos na execução da solicitação “GetProduct”.



(b) O traço visualizado como um gráfico de Gantt e como um gráfico direcionado

Figura 5: Um exemplo de rastreamento para uma solicitação HTTP

Geralmente, há duas etapas para acessar um rastreamento específico: (i) findTraces: um usuário pesquisa rastreamentos que correspondem a um determinado critério (por exemplo, solicitações HTTP desde ontem com mais de 200 ms , nas quais uma chamada de banco de dados retornou pelo menos 2 linhas); (ii) getTrace: na lista de rastreamentos retornados, um usuário seleciona um rastreamento específico para visualizar como um gráfico de Gantt.

Embora o rastreamento tenha sido usado em sistemas distribuídos por décadas [7], há pouca pesquisa sobre o armazenamento e gerenciamento de dados de rastreamento.

No Slack, os volumes de dados de rastreamento são menores do que em outros (por exemplo, 2 TB/dia) devido ao desafio de instrumentar e gerenciar dados de rastreamento. Para gerenciar essa complexidade, o Slack representa os intervalos em um formato mostrado na Figura 5a, semelhante a iniciativas do setor, como a API de rastreamento do OpenTelemetry [14].

2.5 Características comuns dos dados MELT

Conforme discutido nas subseções anteriores e resumido na Tabela 1, os modelos de dados, consulta e armazenamento para os quatro tipos de séries temporais em um ODMS diferem bastante. No entanto, os dados MELT também compartilham diversas características importantes que influenciam como devem ser gerenciados em geral.

Características dos dados. Fundamentalmente, todos os dados MELT são imutáveis e com muitos acréscimos. Além disso, devido ao ambiente distribuído dinâmico, o volume de dados gerados ao longo do tempo é altamente variável (ou seja, em rajadas). Por exemplo, um grande evento de dimensionamento automático ou uma nova exceção de log pode aumentar as taxas de volume de dados de entrada em 10 vezes por um curto período de tempo.

Características da consulta. Uma tendência à atualização resume com precisão a natureza geral das consultas sobre todos os dados de observabilidade. Para ilustrar, a Tabela 2 mostra a distribuição percentual de consultas do Slack por idade dos dados, indicando que >97% de todas as consultas são sobre dados com menos de 24 horas. Durante um incidente, dados atualizados são necessários não apenas para entender o estado atual do sistema, mas também para verificar rapidamente se a correção está tendo sucesso.

Registros de idade de dados (19,6 milhões) Métricas (17 milhões) Rastros (46 mil)			
<1 hora	92,5	94,7	85,2
<2 horas	92,6	95,2	94,8
<4 horas	94,5	97,5	95,0
<1 dia	99,8	99,8	97,3

Tabela 2: % de consultas por idade dos dados (com volumes totais de consultas entre parênteses) para estatísticas de consulta de 1+ meses sobre logs, métricas e rastros no Slack. Mais de 97% de as consultas são para dados produzidos nas últimas 24 horas.

efeito pretendido. Isso implica que novos dados normalmente precisam ser mais acessíveis e disponíveis do que dados mais antigos. Além disso, durante grandes incidentes, não é incomum ter o dobro de número normal de usuários interagindo com os painéis e escrever consultas personalizadas em dados MELT. A maioria das consultas são para painéis e alertas que normalmente consultam apenas um pequeno porcentagem dos dados coletados, mas também exigem subsegundos latências para apoiar a tomada de decisões em tempo real [15].

Gestão do ciclo de vida. Juntamente com a tendência para o frescor, dados históricos ainda são necessários para uma porcentagem menor de consultas ao procurar tendências de longo prazo ou para fins jurídicos/comerciais fins. Por exemplo, durante a interrupção mencionada, Os engenheiros do Slack analisaram os dados das últimas semanas para verificar quaisquer padrões sazonais. Portanto, os dados MELT geralmente requerem gerenciamento do ciclo de vida dos dados para dados novos e dados históricos lado a lado, indicando que um armazenamento em camadas uma estratégia baseada na idade dos dados deve ser projetada no ODMS.

3 DESAFIOS DE HOJE

As soluções ODMS atuais implantadas na indústria são abordagens personalizadas que atendem apenas parcialmente aos requisitos de gerenciamento de dados MELT. Esses sistemas são construídos como resultado de implementação reativa [13], sem projeto orientador princípios e, portanto, enfrentam desafios práticos semelhantes. Em nesta seção, apresentamos o sistema de observabilidade atual do Slack infraestrutura como um estudo de caso para revelar os principais desafios práticos a serem considerados ao arquitetar um ODMS escalável.

3.1 Observabilidade no Slack

A Figura 6 descreve a infraestrutura atual do sistema de observabilidade do Slack . Como muitas soluções do setor, ela consiste em vários ferramentas, pipelines de ingestão e mecanismos de armazenamento para coletar, armazenar, e servir dados MELT gerados por aplicativos Slack.

O Slack usa o Prometheus [20], um sistema pull de nó único , para armazenar e servir métricas. Ele extrai dados de métricas de Pontos de extremidade HTTP expostos por aplicativos Slack. Para alta disponibilidade, o Slack usa um par de servidores Prometheus, cada mantendo uma cópia independente dos dados. Cada aplicação recebe um pool de 2 a 64 servidores Prometheus. No total, O Slack administra cerca de 100 desses pools.

Eventos, logs e rastreamentos (ELT) são enviados ao Apache Kafka. Para durabilidade, Secor [19] consome e grava os dados brutos

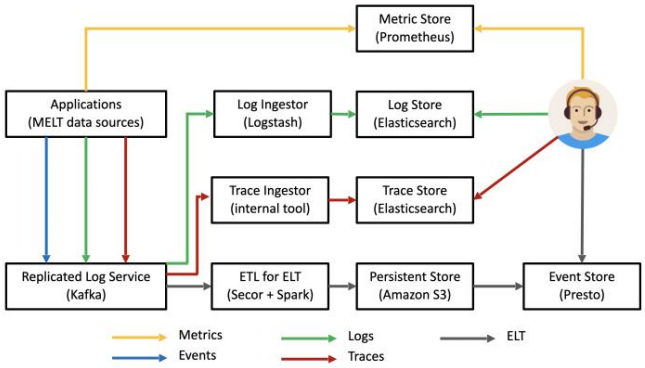


Figura 6: Na infraestrutura de observabilidade atual do Slack, um engenheiro gerencia e consulta quatro sistemas independentes, usando diferentes APIs e ferramentas de consulta.

para o Amazon S3. Um trabalho personalizado do Apache Spark copia e transforma esses dados em um arquivo Parquet colunar, também armazenado em Amazon S3 [26, 28]. Esses arquivos Parquet podem ser consultados via SQL usando Presto [22]. Além disso, o Slack usa Logstash para ingerir logs no Elasticsearch e uma ferramenta interna para ingerir rastreia em uma solução de rastreamento fornecida e um rastreamento interno loja apoiada pelo Elasticsearch [4]. Os logs são retidos por 7 dias e rastros por 14 dias. Os logs e rastros também são gravados em um data warehouse (Presto) para consultas históricas.

3.2 Desafios Práticos

A arquitetura ODMS atual do Slack enfrenta uma série de desafios práticos exclusivos para os requisitos de crescentes cargas de trabalho de observabilidade. Embora usemos o Slack como um estudo de caso, Acreditamos que quase todos os ODMSs atualmente implantados na indústria enfrentam um ou mais desses desafios práticos. Agrupamos estes desafios em três categorias abrangentes:

Alta Complexidade Operacional. A infraestrutura para atender Os dados MELT no Slack contêm mais de 20 componentes de software separados . Cada componente tem uma arquitetura única e, como como resultado, precisa de operações personalizadas para gerenciamento de cluster, segurança e planejamento de capacidade. Essa heterogeneidade leva a soluções complexas. Por exemplo, Prometheus é um nó único sistema e mantém cópias independentes dos dados para atender requisitos de alta disponibilidade. Enquanto isso, embora o Elasticsearch tenha gerenciamento de réplica integrado, na escala do Slack, ele é desafiador de gerenciar e operar. Durante o incidente de 12 de maio , o volume de dados atingiu um pico de 4 vezes o volume normal. Em tais cenários, a equipe de monitoramento realiza tarefas complexas operações de ampliação para continuar ingerindo novos dados de telemetria.

Mantendo Baixa Latência de Consultas. Consultas sobre Observabilidade os dados são altamente distorcidos. Mais de 97% das consultas acessam dados < 24 horas para alertas quase em tempo real. Painéis e grandes incidentes ou mudanças de produtos resultam em picos significativos no número de consultas que colocam uma pressão significativa sobre o ODMS. Durante o incidente de 12 de maio, o Slack analisou os dados

nas 8 horas anteriores, enquanto lidava com o dobro da carga de pico histórica. A alta complexidade operacional do ODMS do Slack torna o dimensionamento em resposta à carga de trabalho geral de consultas e a manutenção de baixa latência de consulta um desafio significativo.

Alto Custo de Infraestrutura. Na escala de petabytes, atender aos crescentes requisitos de retenção de dados e garantir a disponibilidade dos dados em cargas de trabalho intensas rapidamente se torna custoso. No Slack, um tempo considerável é gasto equilibrando o custo de infraestrutura , desempenho, durabilidade e disponibilidade dos dados.

Por exemplo, o Slack duplica o processamento ELT no Presto para disponibilidade e durabilidade, correndo o risco de desempenho mais lento e custos de infraestrutura mais altos. O Slack também provisiona com base em picos históricos para lidar com cargas de trabalho com picos, como durante o incidente de 12 de maio. O novo pico de carga agora é 2 vezes maior que o pico de carga anterior ao incidente. Essa estratégia de provisionamento aumenta o custo da infraestrutura.

4 PRINCÍPIOS DE DESIGN DE ODMS

Agora propomos um conjunto de princípios de design que abordam os requisitos e desafios do mundo real descritos nas seções anteriores. Considerando a natureza heterogênea dos dados MELT e a necessidade de reduzir a complexidade do gerenciamento desses dados enquanto atende aos requisitos de desempenho de consulta de forma escalável, acreditamos que um ODMS deve aderir a quatro princípios de design principais: **Desacopla o gerenciamento de dados em tempo real e histórico.** A §2 e a Tabela 2 mostraram que a carga de trabalho do ODMS é significativamente diferente das cargas de trabalho de banco de dados híbrido que combinam OLAP e OLTP (por exemplo, HTAP [16]). Ele ingere petabytes de gravações imutáveis potencialmente intermitentes e as consultas são tendenciosas para dados com menos de 24 horas. Os ODMSs projetados com essa carga de trabalho em mente desacopla o gerenciamento de dados em tempo real e histórico. Esse desacoplamento mantém taxas de ingestão rápidas, baixa latência de consulta em dados recentes e minimiza o custo de armazenamento e acesso a dados históricos.

Unifique a gestão do ciclo de vida dos dados MELT. O tempo vincula toda a gestão de dados MELT a uma estrutura comum. As estratégias exclusivas de gestão de dados históricos e em tempo real devem ser regidas por um ciclo de vida unificado, baseado na idade dos dados. Isso deve abstrair a movimentação de dados do armazenamento em tempo real para o histórico, reduzindo o custo e a complexidade do acesso transparente a todos os dados em períodos arbitrários.

Forneça uma interface de consulta única para dados MELT. Consultas de observabilidade geralmente requerem dados de diferentes tipos de MELT. Uma única interface de consulta abstrai os requisitos individuais de armazenamento e processamento dos dados MELT, diminuindo a complexidade da escrita dessas consultas e proporcionando oportunidades de otimização em todas as estratégias de armazenamento. Este princípio se presta a uma arquitetura de mecanismo de armazenamento plugável "semelhante a um polystore" [3].

Suporte à implantação nativa e distribuída da nuvem. Devido à natureza altamente intermitente de leituras e gravações, várias camadas

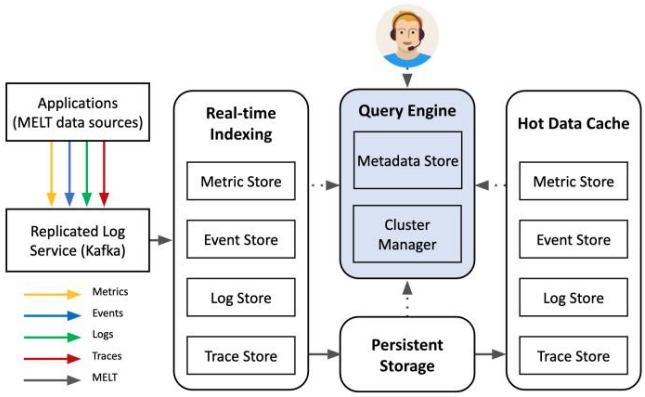


Figura 7: Na nova arquitetura baseada nos princípios de design do ODMS, um usuário gerencia e consulta um único sistema.

O sistema deve ser distribuído e elástico por design, para que o sistema possa ser escalável com a carga de trabalho em constante mudança. Um ODMS nativo da nuvem permite que a equipe de observabilidade faça um trade-off flexível entre custo e desempenho para fornecer dados MELT com base na carga de trabalho e no ciclo de vida dos dados.

Esses quatro princípios devem ser integrados ao processo de design de um ODMS. Diferentemente da prática atual do setor, em que os ODMSs são construídos como uma colcha de retalhos de recursos que atendem a requisitos ad hoc, seguir esses princípios resulta em uma arquitetura coerente que proporciona controle sobre a complexidade, o desempenho e o custo de um ODMS.

A Figura 7 mostra uma visão geral de uma arquitetura que concretiza esses princípios de design. Ela unifica o ciclo de vida dos dados MELT em uma única infraestrutura ODMS. Ao contrário do sistema atual do Slack, que é influenciado por arquiteturas Lambda modernas [5], este design é influenciado pelo design de arquiteturas polystore modernas [3]. Isso reduz os requisitos de armazenamento e a complexidade da manutenção de múltiplos pipelines de dados e atende à necessidade de coordenação transparente de múltiplos mecanismos de armazenamento.

Ingestão e armazenamento de dados. Todos os dados de entrada MELT de aplicativos instrumentados são ingeridos em um Serviço de Log Replicado (por exemplo, Apache Kafka [6]). A camada de Indexação em Tempo Real, composta por mecanismos de armazenamento específicos para cada tipo de dado, extrai dados do serviço de log e os indexa para acesso rápido.

Periodicamente, os arquivos de dados indexados são migrados para a camada de Armazenamento Persistente, semelhante ao Amazon S3. Essa migração é coordenada pelo Gerenciador de Cluster. No momento da consulta, o Mecanismo de Consulta coordena as camadas de Indexação em Tempo Real e Armazenamento Persistente usando o Armazenamento de Metadados. Quando necessário, o Mecanismo de Consulta extrai dados históricos importantes da camada de Armazenamento Persistente para o Cache de Dados Importantes para manter o desempenho da consulta .

Processamento de consultas. O Mecanismo de Consulta atende a todas as consultas. Usando o Metadata Store, ele coordena entre as três camadas de dados (indexação em tempo real, armazenamento persistente e hot

Cache de Dados) para filtrar dados com base no intervalo de tempo e otimizar consultas em vários tipos de dados (semelhante a junções por tempo). Ele determina o posicionamento ideal dos dados com base nas distribuições e custos esperados das consultas. Por exemplo, consultas ad-hoc que acessam dados no Armazenamento Persistente podem não precisar ser armazenadas em cache. No entanto, ao acessar dados históricos repetidamente durante a resolução de um problema, o mecanismo de consulta pode decidir copiar dados para o Hot Data Cache para minimizar a latência da consulta.

Distribuição e disponibilidade. A estratégia de distribuição do ODMS deve manter alta disponibilidade durante períodos de pico de novos dados e durante períodos com carga de consulta particularmente pesada. As camadas de Indexação em Tempo Real e Cache de Dados Quentes são nativamente elásticas, onde réplicas independentes são geradas sob demanda. O Gerenciador de Cluster do Mecanismo de Consulta monitora a carga de trabalho da camada de Indexação em Tempo Real para determinar se deve aumentar a escala de algum dos armazenamentos de componentes. Durante períodos de gravações de dados particularmente pesadas (por exemplo, de aplicativos instrumentados) ou consultas em dados novos, ele aumenta a escala da camada de Indexação em Tempo Real. Durante leituras particularmente pesadas de dados históricos, ele aumenta a escala do Cache de Dados Quentes. O design de duas camadas também ajuda a suportar a disponibilidade variável de dados (por exemplo, garantias de maior disponibilidade para dados mais recentes/valiosos).

A arquitetura pode ser generalizada para cenários onde a telemetria de observabilidade é unificada em um ODMS centralizado que torna os dados MELT facilmente acessíveis aos usuários. Uma arquitetura ODMS que siga esses princípios visa auxiliar durante picos de carga, como o ocorrido durante o incidente do Slack em 12 de maio.

O **Mecanismo de Consulta** fornece uma interface de consulta única e uma visão unificada dos dados MELT. Ele fornece uma abstração sobre a complexidade da consulta e do gerenciamento dos tipos heterogêneos e seus ciclos de vida nas camadas **de Indexação em Tempo Real** e **Armazenamento Persistente** histórico . O **Hot Data Cache** elástico responde a picos de carga com base em padrões de acesso a dados para manter baixa latência de consulta. Por fim, o **Gerenciador de Cluster** gerencia os ciclos de vida dos dados e a complexidade operacional neste sistema. Ele também dimensiona ou reduz com base na carga de trabalho, reduzindo assim o custo geral da infraestrutura do sistema . Esses princípios resultam em menor tempo para obter insights durante picos de carga e reduzem a complexidade e os custos gerais.

5 CONCLUSÃO

O gerenciamento de dados de observabilidade é uma área emergente de pesquisa que requer mais atenção da comunidade de bancos de dados. Neste artigo, discutimos a experiência real com dados de observabilidade e seus casos de uso no Slack – um serviço de colaboração em equipe baseado em nuvem. A natureza heterogênea dos dados de séries temporais, bem como as características variáveis da carga de trabalho, exigem uma nova arquitetura de gerenciamento de dados de observabilidade. Em resposta, propusemos uma nova arquitetura nativa em nuvem, semelhante a um polystore, que desacopla as camadas de acesso a dados históricos e em tempo real do armazenamento persistente subjacente e da camada de consulta, de forma a permitir seu dimensionamento independente.

estão atualmente trabalhando na construção de um protótipo inicial deste design para testar com dados de produção do Slack.

REFERÊNCIAS

[1] RH Arpaci-Dusseau et al. 2018. Sistemas de arquivos nativos da nuvem. Na Conferência USENIX sobre tópicos importantes em computação em nuvem (HotCloud).

[2] C. Chan et al. 2020. Incidentes de depuração no Google Distribuído Sistemas. ACM Fila 18, 2 (2020).

[3] J. Duggan et al. 2015. O sistema de armazenamento de polígonos BigDAWG. ACM SIG-MOD Record 44, 2 (2015), 11–16.

[4] C. Gormley et al. 2015. Elasticsearch: O Guia Definitivo. O'Reilly Mídia.

[5] M. Hausenblas et al. 2017. Arquitetura Lambda. http://lambda-arquitetura.net.

[6] N. Narkhede et al. 2017. Kafka: O Guia Definitivo Dados em Tempo Real e Processamento de Fluxo em Escala. O'Reilly Media.

[7] J. Jeffrey et al. 1987. Monitoramento de sistemas distribuídos. ACM Transactions on Computer Systems (TOCS) 5, 2 (1987), 121–150.

[8] J. Kaldor et al. 2017. Canopy: Um rastreamento de desempenho de ponta a ponta E Sistema de Análise. Em SOSP. 34–50.

[9] R. Katkov. 2020. Todos a postos. https://slack.engineering/all-hands-on-deck-91d6986c3ee .

[10] L. Lamport. 1976. A ordenação de eventos em um sistema distribuído. Comunicações da ACM 21, 7 (1976), 558.

[11] J. Mace et al. 2015. Rastreamento de pivô: monitoramento causal dinâmico para Sistemas Distribuídos. Em SOSP. 378–393.

[12] S. More. 2018. Um guia prático de observabilidade. mStakx.

[13] S. Niedermaier et al. 2019. Sobre a observabilidade e o monitoramento de dis-Sistemas Atribuídos – Um Estudo de Entrevistas na Indústria. No ICSOC.

[14] OpenTelemetry. 2019. A observabilidade de código aberto OpenTelemetry Estrutura. https://opentelemetry.io/.

[15] J. O'Shea. 2020. Construindo painéis para visibilidade operacional . https:// aws.amazon.com/builders-library/building-dashboards-for- operational-visibility/.

[16] F. Özcan et al. 2017. Processamento transacional/analítico híbrido: uma pesquisa. Na conferência ACM SIGMOD. 1771–1775.

[17] T. Palpanas. 2015. Gerenciamento de séries de dados: o caminho para a análise de grandes sequências. ACM SIGMOD Record 44, 2 (2015), 47–52.

[18] T. Palpanas et al. 2019. Relatório sobre o Primeiro e Segundo Workshops Interdisciplinares de Análise de Séries Temporais. ACM SIGMOD Record 48, 3 (2019), 36–40.

[19] Pinterest. 2017. Pinterest Secor: Um serviço para implementar persistência de log do Kafka. https://github.com/pinterest/secor.

[20] Prometheus. 2012. Documentação do Prometheus. https://prometheus.io/ docs/concepts/ metric_types/.

[21] J. Rodrigues et al. 2017. Sieve: Insights acionáveis a partir de métricas monitoradas em sistemas distribuídos. Em ACM Middleware Conference. 14–27 .

[22] R. Sethi et al. 2019. Presto: SQL em tudo. Em IEEE ICDE.

[23] Y. Shkuro. 2019. Dominando o rastreamento distribuído: analisando o desempenho em microsserviços e sistemas complexos. Packt Publishing.

[24] BH Sigelman et al. 2010. Dapper: Uma infraestrutura de rastreamento de sistemas distribuídos em larga escala . Relatório técnico. Google, Inc.

[25] C. Sridharan. 2018. Observabilidade de sistemas distribuídos: um guia para a construção de sistemas robustos. O'Reilly Media.

[26] D. Vohra. 2016. Apache Parquet. Em Ecossistema Hadoop prático: um guia definitivo para estruturas e ferramentas relacionadas ao Hadoop. 325–335.

[27] A. Wiedemann et al. 2019. O fenômeno DevOps. Fila ACM 17, 2 (2019).

[28] M. Zaharia et al. 2010. Spark: Computação em cluster com conjuntos de trabalho. Na Conferência USENIX sobre tópicos importantes em computação em nuvem (HotCloud).