Принципы построения высоконагруженных систем
Институт прикладных компьютерных наук ИТМО

Домашнее задание 1. Мониторинг и нагрузочное тестирование

Георгий Семенов
georgii.v.semenov@mail.ru
Мягкий дедлайн: Сб, 22.11.2025, 23:59 МСК
Жесткий дедлайн: Сб, 29.11.2025, 23:59 МСК

Данила Красов
DDIA25-HW1-DanilaKrasov.pdf

November 29, 2025

Правила курса

- Задание выдается на 2 недели с двумя дедлайнами:
  - **Мягкий дедлайн** – в рамках этого периода можно заранее отправить решение и получить обратную связь, чтобы исправить замечания до наступления жесткого дедлайна.
  - **Жесткий дедлайн** – после этого новые посылки работы не принимаются, сдать работу больше нельзя.
- Планируется выдать 4 домашних задания, каждое из которых стоит ±12 баллов. Правила выставления оценки за курс следующие:
  - **A** – **«5»** – $\geq 90\%$ от общей суммы обязательных баллов (предв. ≥ 43.2)
  - **B/C** – **«4»** – $\geq 75\%$ от общей суммы обязательных баллов (предв. ≥ 36)
  - **D/E** – **«3»** – $\geq 60\%$ от общей суммы обязательных баллов (предв. ≥ 28.8)
  - **F** – **«2»** – $< 60\%$ от общей суммы обязательных баллов (предв. < 28.8)
- Выполненное задание рекомендуется оформить в LaTeX(например, в Overleaf) и выслать файлом с именем вида `DDIA25-HW1-IvanIvanov.pdf` на почту выше.
  - Пожалуйста, оформляйте ответы на задания в блоке `Решение`.

Задание	1.1	1.2	1.3	1.4 (*)	2.1	2.2	Σ	Σ (*)
Баллы	1.5	2.5	3	2	1.5	3.5	12	14

1 Мониторинг

В ходе выполнения заданий в этом разделе вам предстоит помочь корпорации *Монокль* внедрить актуальные стандарты мониторинга на основе Prometheus.

1.1 Метрики и их классы

Ранее в *Монокле* у каждого бизнес-юнита были свои политики мониторинга и надежности, наиболее подходящие его продуктам и сервисам. Настал момент перейти к централизованной системе мониторинга и внедрить единые стандарты телеметрии, в т.ч. сбора метрик.

Задание: (1.5 б.) Помогите SRE-команде *Монокля* сформировать глоссарий терминов для нового корпоративного стандарта *observability*. Заполните таблицы ниже.

Решение:

«Observability Pillar»	Характеристика
Metrics	Числовые показатели состояния системы во времени: latencies и throughput, error rates - для агрегирования и алертинга.
Logs	Структурированные или неструктурированные события с подробным контекстом для расследования инцидентов и отладки системы.
Traces	Распределённые трейсы для корреляции запросов между сервисами и расследования инцидентов.
Events/Topology	События развёртывания, конфигурации и данные о связях между компонентами. Важны для интерпретации метрик и трасс.

«Golden Signal»	Пример метрики	Единица измерения
Latency	Время обработки HTTP-запроса веб-сервером. Меряется для перцентилей пользователей.	миллисекунды (ms)
Traffic	Количество HTTP-запросов, обрабатываемых веб-сервером в секунду.	запросы в секунду (rps)
Errors	Доля неуспешных HTTP-запросов (5xx) от общего числа запросов к веб-серверу.	процент (%)
Saturation	Процент использования CPU веб-сервера от его полной мощности.	процент (%)

Метрика доступности	Описание
Uptime (%)	Доля времени, когда система была доступна. ИМХО, сложно объективно оценить доступность, так как система редко выходит из строя полностью. Однако, как показала практика Amazon, выход строя критических компонентов делает недоступной всю систему целиком.
Downtime (s)	Время, когда система была недоступна.
MTBF (s)	Среднее время между отказами системы.
MTTR (s)	Среднее время восстановления после отказа.
MTTD (s)	Среднее время обнаружения отказа.

□

1.2 SLA

Бизнес-юнит *Монолит* сейчас использует собственную систему мониторинга *Монада* со своим форматом хранения метрик. Метрики *Монады* хранятся в иерархической структуре (например, `//monolith/web/http_requests/timings`) и соответствуют лишь двум типам: RPS-счётчики (`rate`) и перцентили (`percentile`). Поддерживаются только перцентили из множества $(0.5, 0.75, 0.9, 0.95, 0.99)$.

Настало время «распилить *Монолит*» и канонично присоединиться к прогрессивному миру PromQL. SRE-команда *БЮ Монолит* добродушно реализовала PromQL-адаптер к своей системе мониторинга. С ним можно выбрать метрику *Монады* с помощью метки `monad`, указав путь `path`, облачную зону `cloud_dc` (поддерживаются зоны `AB, BC, CA`), тип поля и метрики `field` и `type` (что бы эта идиома ни значила).

Стажер в SRE-команде *Монокля* получил задачу реализовать централизованный мониторинг SLO на основе предоставленного PromQL-адаптера для бэкенда *Монолита* – необходимо отслеживать 99-ый перцентиль времени ответа и суммарный RPS запросов:

```
avg (monad{
        cloud_dc=~".*",
        path="//monolith/web/http_requests/timings",
        field="p99",
        type="quantile"
})

sum (monad{
        cloud_dc=~".*",
        path="//monolith/web/http_requests/requests",
        field="1",
        type="rate"
})
```

Задание: Помогите SRE-команде *Монокля* оценить и обосновать корректность результата их стажера.

1. (0.75 б.) Приведите пример инцидента, при котором метрики стажера не зафиксируют реальную деградацию доступности сервиса для $> 1\%$ пользователей *Монолита*.

2. (0.75 б.) Поможет ли использование взвешенного среднего вместо простого усреднения по зонам избежать проблемы в пункте выше? Проиллюстрируйте ответ.

3. (0.5 б.) Предложите, как можно усовершенствовать предоставленный PromQL-адаптер, чтобы помочь стажеру правильно составить необходимые запросы.

4. (0.5 б.) Запишите корректные PromQL-выражения для 99-го перцентиля времени ответа и суммарного RPS запросов в предположении, что эти метрики изначально были доступны как «настоящие» Prometheus-метрики, а не как адаптер через *label* с именем `monad`.

Решение:

1. Проблема в том, что усреднение перцентилей не даёт корректной оценки перцентиля по совокупности пользователей. Например, если в зоне AB 98% пользователей получают время ответа 100 мс, а 2% - 1000 мс, то 99-й перцентиль в этой зоне будет 1000 мс. Допустим, в остальных зонах все пользователи получают 100 мс. Тогда усреднённый 99-й перцентиль по всем зонам будет $(1000 + 100 + 100)/3 = 400$ мс, что не только не отражает реальную ситуацию, где лишь 2% пользователей одной из зон испытывают деградацию, но и вводит в заблуждение относительно общей производительности сервиса увеличением 99-го перцентиля до 400 мс. 2. Например в зоне AB 20% трафика (то есть вес равен 0.2). Тогда взвешенный 99-й перцентиль будет $(1000 * 0.2 + 100 * 0.8) = 280$ мс, что всё равно не отражает реальную ситуацию, хотя и несколько сглаживает её. 3. Необходимо отказаться от агрегации и вычислять перцентили по зонам отдельно.

```
avg by (cloud_dc) (monad{
        cloud_dc=~".*",
        path="//monolith/web/http_requests/timings",
        field="p99",
        type="quantile"
})

sum by (cloud_dc) (monad{
        cloud_dc=~".*",
        path="//monolith/web/http_requests/requests",
        field="1",
        type="rate"
})
```

4. PromQL-выражения для 99-го перцентиля времени ответа и суммарного RPS запросов:

```
histogram_quantile(0.99, sum by (le) (rate(http_request_duration_seconds

sum(rate(http_requests_total[5m]))
```

□

1.3 Светофорные дашборды

Разобравшись с деталями интеграции бизнес-юнитов корпорации, SRE-команда *Монокля* принялась за создание единой 24/7 дежурной смены и дашбордов для мониторинга всех сервисов корпорации. Чтобы дежурный мог быстро оценить состояние систем, главный дашборд в Grafana должен быть оформлен как множество *светофорных плиток* – каждая плитка Stat отображает состояние одного сервиса:

- Зеленая плитка OK означает, что все SLO сервиса выполняются.
- Желтая плитка WARN означает, что какой-то SLO сервиса находится в состоянии предупреждения.
- Красная плитка CRIT означает, что какой-то SLO сервиса не выполнен.
- Серая плитка ? означает, что какой-то из SLO не получилось посчитать (например, из-за No Data или ошибки запроса).

SRE-команда сразу же решила не заниматься накликиванием дашбордов вручную, а автоматизировать процесс создания дашбордов на основе шаблонов и API Grafana. Для этого они хотят написать службу, которая конвертирует описание SLO сервисов в красивый светофорный дашборд. Перед этим они разработали некое формальное исчисление для описания SLO сервисов корпорации, в него вошли следующие конструкции:

- $S_i :: service$ – сервис с номером i.
- $const :: scalar$ – можно объявить любую константу.
- $\{\text{UNK}, \text{CRIT}, \text{WARN}, \text{OK}\} :: state$ – монотонно возрастающее перечисление состояний многозначного SLO.
- $T_{99\%}(S_i) :: scalar?$ – время ответа (ms) в текущий момент у сервиса S_i для 99% пользователей.
- $R(S_i) :: scalar?$ – рейт запросов (rps) в текущий момент у сервиса S_i.
- $<:: scalar? \to scalar? \to state$ – оператор сравнения скалярных величин; если одна из величин неизвестна, возвращает UNK; если условие верно, возвращает OK; если условие неверно, возвращает CRIT.
- $\vee :: state \to state \to state$ – оператор объединения условий, возвращающий наилучшее из состояний; если одно из состояний UNK, то возвращает UNK.
- $\wedge :: state \to state \to state$ – оператор объединения условий, возвращающий наихудшее из состояний; если одно из состояний UNK, то возвращает UNK.

Так, корректным термом этого исчисления считается выражение типа $state$, например:

$$(T_{99\%}(S_1) < 200) \wedge (2000 < R(S_1)) \wedge (R(S_1) < 10000) :: state$$

$$(20 < 300) \vee \text{WARN} = \text{OK} :: state$$

Задание: Помогите SRE-команде *Монокля* реализовать транслятор их формализма в плитки Stat для дашборда в Grafana.

1. (0.25 б.) Запишите терм для следующего SLO: «99%-время ответа сервиса S_1 должно быть больше 200 мс для WARN и больше 300 мс для CRIT».

2. (0.5 б.) Отобразите в PromQL типы scalar и state и опишите, с помощью каких свойств Stat в Grafana необходимо раскрасить плитки в зависимости от значения state.

3. (0.75 б.) Покажите, как реализовать операторы $<, \vee, \wedge$ предложенного исчисления в PromQL (подсказка: используйте выражения вида `OR on() vector(0)`).

4. (1.5 б.) Предложите три SLO для параметров вашего ноутбука (docker-окружения), которые можно замерить с помощью `node-exporter`. В окружении Grafana с семинара создайте дашборд из трех светофорных плиток (необязательно Stat, но с раскраской) для них. Прикрепите три терма для ваших SLO, соответствующие им PromQL-выражения и скриншоты дашборда. Один из SLO должен содержать `WARN` уровень.

Решение:

1. $(T_{99\%}(S_1) < 300) \wedge (T_{99\%}(S_1) < 200 \vee \text{WARN}) :: state$

2. В PromQL тип scalar можно отобразить как `vector`, а тип state как `vector` с числовыми значениями: `0` для UNK, `1` для CRIT, `2` для WARN и `3` для OK. В Grafana Stat раскраска задаётся по порогам числового значения state: value=3 $\rightarrow$ зелёный (OK), 2 $\rightarrow$ жёлтый (WARN), 1 $\rightarrow$ красный (CRIT), 0 или отсутствует $\rightarrow$ серый (?).

3. Для реализации операторов $<, \vee, \wedge$ в PromQL можно использовать функции сравнения и объединения, такие как `or`, `and` и `unless`. Например, для оператора $<$ можно использовать выражение $\texttt{T}_{99\%}(S_1) < 200$.$\vee$ `можно использовать state1 or state2, а для оператора` $\wedge$ `- state1 and state2.`

4. `SLO для ноутбука:`

   - `SLO 1: CPU usage should be less than 80% for WARN and less than 90% for CRIT.`
     - `Терм:` $(CPU\_usage < 90) \wedge (CPU\_usage < 80 \vee \texttt{WARN}) :: state$
     - `PromQL: 100 - (avg by(instance) (rate(node_cpu_seconds_total{mode="idle"}[5m * 100)`
   - `SLO 2: Memory usage should be less than 70% for WARN and less than 85% for CRIT.`

  - Term: $(Memory\_usage < 85) \wedge (Memory\_usage < 70 \vee \texttt{WARN}) :: state$
  - PromQL: `(node_memory_MemTotal_bytes - node_memory_MemAvailable_bytes) / node_memory_MemTotal_bytes * 100`
- SLO 3: Disk I/O wait time should be less than 5ms for OK.
  - Term: $(Disk\_io\_wait < 5) :: state$
  - PromQL: `rate(node_disk_io_time_seconds_total[5m]) * 1000`

□

1.4 Recording rules (*)

SRE-команда *Монокля* доказала свою компетентность в светофорных дашбордах, поразила своими скиллами все продуктовые команды и уже почти обрадовала топ-менеджмент. Но на демонстрации произошла неловкость.

На светофорной борде с интервалом $d = 5s$ было отображено более $n = 200$ светофорчиков, каждый из которых был реализован сложным PromQL-запросом с множеством операторов, рассмотренных выше. Prometheus не справлялся с этой нагрузкой, а Grafana подвисала при попытке отобразить дашборд. Топ-менеджмент был расстроен.

SRE-команда *Монокля* решила исправить ситуацию с помощью *recording rules*.

Задание: [1]

- (0.5 б.) Объясните, как использование *recording rules* поможет SRE-команде *Монокля* в данной ситуации.
- (1 б.) Напишите *rule group* для ваших метрик светофорных плиток из предыдущего задания в `prometheus.yml`. Проверьте корректность вашего файла с помощью `promtool` и продемонстрируйте результат. Оцените, насколько это субъективно ускорило загрузку вашего дашборда при малом интервале обновления.
- (0.5 б.) Какие ограничения/недостатки у использования *recording rules* вы можете назвать?

Решение:

- Использование recording rules позволяет заранее вычислить и сохранить результаты сложных PromQL-запросов в виде новых временных рядов. Это снижает нагрузку на Prometheus при выполнении запросов в реальном времени, так как вместо повторного вычисления сложных выражений, система может просто обращаться к уже сохранённым результатам.
- Пример rule group для метрик светофорных плиток: ...
- Ограничения/недостатки использования recording rules:
  - Результаты записываются с определённым интервалом, что может привести к устаревшим данным.
  - Сохранение дополнительных временных рядов требует больше дискового пространства.
  - Необходимо поддерживать актуальность и корректность записанных правил.

□

[1] Задания и пункты, помеченные звездочкой, не являются обязательными. Эти баллы учитываются в общей сумме за курс, но являются дополнительными. Иными словами, вы можете сделать дополнительное задание вместо какого-то обязательного и получить тот же суммарный балл.

2 Нагрузочное тестирование

2.1 Закон Амдала

Предположим, что программа выполняет операцию широковещательной рассылки (broadcast). Эта операция вносит дополнительное время выполнения, зависящее от числа задействованных ядер n. Доступны две реализации broadcast:

- первая добавляет параллельные накладные расходы $\beta = 0.0001n$;
- вторая — $\beta = 0.0005 \cdot \ln(n)$.

Задание: (1.5 б.) Для какого числа ядер достигается наибольшее ускорение программы с долей последовательного кода$\alpha = 0.001$ в каждой из реализаций?

Решение:

Решение: Согласно закону Амдала, ускорение $S(n)$ программы при использовании n ядер вычисляется по формуле:

$$S(n) = \frac{1}{(1-p) + \frac{p}{n} + \beta(n)}$$

где p - доля параллельного кода, а $\beta(n)$ - накладные расходы на параллелизм. В нашем случае, доля последовательного кода $\alpha = 0.001$, следовательно, доля параллельного кода $p = 1 - \alpha = 0.999$. 1. Для первой реализации с $\beta(n) = 0.0001n$:

$$S_1(n) = \frac{1}{0.001 + \frac{0.999}{n} + 0.0001n}$$

Чтобы найти максимальное ускорение, нужно найти производную $S_1(n)$ по n и приравнять её к нулю:

$$\frac{dS_1}{dn} = 0$$

Решая это уравнение, получаем оптимальное число ядер $n_1 = 100$.

2. Для второй реализации с $\beta(n) = 0.0005 \cdot \ln(n)$:

$$S_2(n) = \frac{1}{0.001 + \frac{0.999}{n} + 0.0005 \cdot \ln(n)}$$

Аналогично, находим производную $S_2(n)$ по n и приравниваем её к нулю:

$$\frac{dS_2}{dn} = 0$$

Решая это уравнение, получаем оптимальное число ядер $n_2 \approx 2000$.

2.2 Case study

В рамках этого задания вам предлагается выбрать систему, рассмотреть ее архитектуру, предложить оценку нагрузки и SLA, выявить *bottlenecks* (см. лекцию) и предложить метрики для мониторинга (см. задание 1.1).

Задание:

1. (1 б.) Опишите архитектуру и компоненты выбранной вами системы. Оцените внешнюю входную нагрузку и гарантии: throughput, latency, количество пользователей, объем хранимых данных. Прокомментируйте ваши оценки.

2. (1.25 б.) Выявите 3-5 потенциальных *bottlenecks* – проспекулируйте, в каких частях системы они могут возникнуть? Рассмотрите отдельно *database bottlenecks*.

3. (1.25 б.) Предложите метрики и их SLO для покрытия предложенных вами *bottlenecks*. Какие стратегии автоматического реагирования можно предложить в случае срабатывания алертов на них?

Решение:

Выбранная система: Twitter (социальная сеть для микроблогов и обмена сообщениями).

1. Архитектура и компоненты:

- Клиентские приложения (веб, мобильные).
- Load Balancer для распределения трафика.
- Api-Gateway для распределения запросов по микросервисам.
- Микросервисы для обработки твитов, сообщений, уведомлений и т.д. (сердце системы).
- Базы данных (SQL и NoSQL) для хранения твитов, пользователей, подписок и т.д.
- Кэширование (Redis, Memcached) для ускорения доступа к часто запрашиваемым данным.
- CDN для ускоренного доступа к статическому медиа-контенту.
- Системы очередей (Kafka, RabbitMQ) для обработки асинхронной обработки событий.
- Системы мониторинга и логирования (Prometheus, Grafana, ELK Stack и т.д.).

Оценка нагрузки и гарантии:

- Throughput: Обработка миллионов твитов и сообщений в секунду (например, 10 млн твитов/сек).
- Latency: Среднее время отклика API должно быть менее 200 мс для 99% запросов.

- Количество пользователей: Более 300 миллионов активных пользователей в месяц.
- Объем хранимых данных: Более 500 ТБ текстовых данных и более 1 ПБ медиа-контента.

Комментарий: Эти оценки основаны на публично доступной информации о Twitter и типичных требованиях к высоконагруженным системам социальных сетей.

2. Потенциальные bottlenecks:

- При отказе узла с кешем может возникнуть спайковая нагрузка на базу данных, что в лучшем случае приведет к увеличению задержек, а в худшем – к отказу базы данных.
- Поиск твитов по самым популярным хэштегам может создать узкое место в сервисах поиска и индексации, так как запросы будут обрабатываться одними и теми же серверами (горячие ключи).
- При посте популярного твита может возникнуть нагрузочный пик на сервисы обработки сообщений. База данных может стать узким местом при массовом чтении/записи одного и того же твита.
- При высокой нагрузке на запись (например, лайки, ретвиты) между репликами вероятно возникет высокий лаг репликации, что приведет к несогласованности данных. Пользователь, отправивший твит может не увидеть его сразу в своей ленте.

3. Метрики и SLO:

- Отказ узла с кешем:
  - Метрика: Cache Hit Rate
  - SLO: Cache Hit Rate должен быть не менее 95% в течение 15 минут.
  - Реагирование: перенаправление трафика на резервные узлы кеша, восстановление отказавших узлов.
- Поиск твитов по популярным хэштегам:
  - Метрика: Latency of Search Queries
  - SLO: 99-й перцентиль Latency должен быть менее 300 мс в течение 5 минут.
  - Реагирование: масштабирование сервисов поиска.
- Обработка популярных твитов:
  - Метрика: CPU Usage of Message Processing Services
  - SLO: CPU Usage должен быть менее 80% в течение 10 минут.
  - Реагирование: разворачиваение дополнительных инстансов сервисов обработки сообщений.

- Лаг репликации базы данных:
  - Метрика: Replication Lag
  - SLO: Replication Lag должен быть менее 5 секунд в течение 10 минут.
  - Реагирование: уведомление администраторов базы данных для ручного вмешательства, масштабирование базы данных (если возможно).

□