



Metrics at Full Throttle: *Intro and Deep Dive into* ***Thanos***

Hello!

Filip Petkovski

- Staff Production Engineer @ Shopify
- [Thanos](#) Maintainer
- [@fpetkovski](#) on Twitter/GitHub
- Based in Munich, Germany



Hello!

Saswata Mukherjee

- Software Engineer @ Red Hat
- [Thanos](#), [Observatorium](#), [mdox](#), other Go [lib](#) Maintainer
- GSoC'21 [Thanos](#) mentee
- [@saswatamcode](#) on Twitter/GitHub/elsewhere



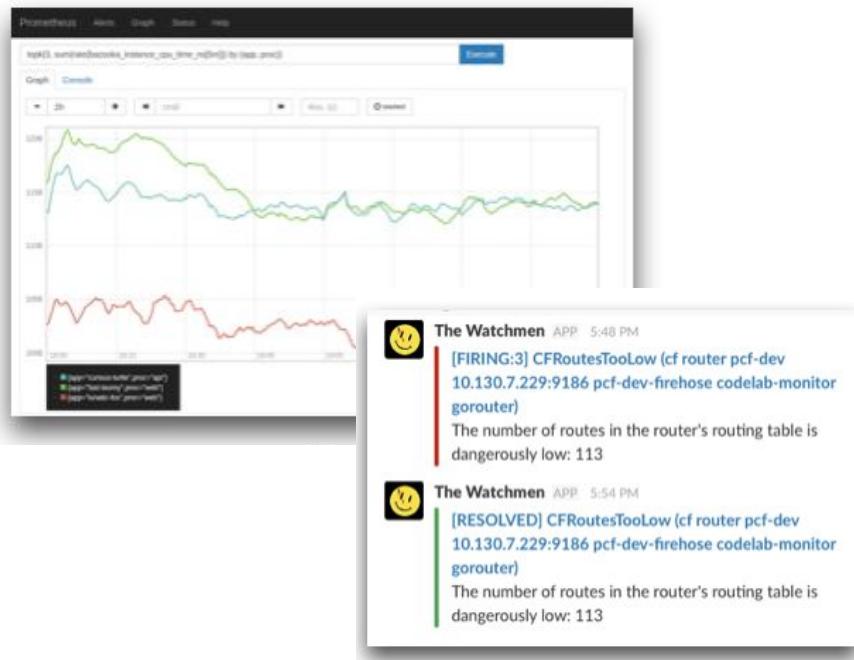
Prometheus!

Prometheus



Prometheus

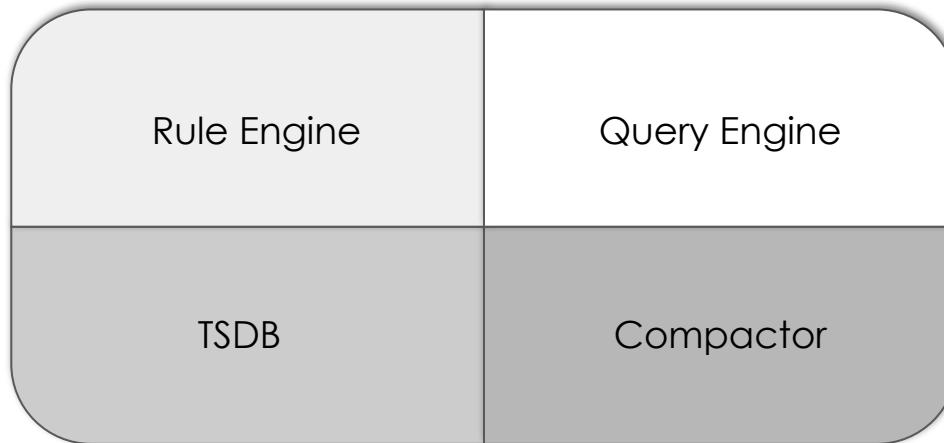
- Standalone monitoring server
- Deployed close to applications
- No external dependencies
- Stores data locally
- Flexible query language (PromQL)
- Yells at you when something is wrong



Prometheus

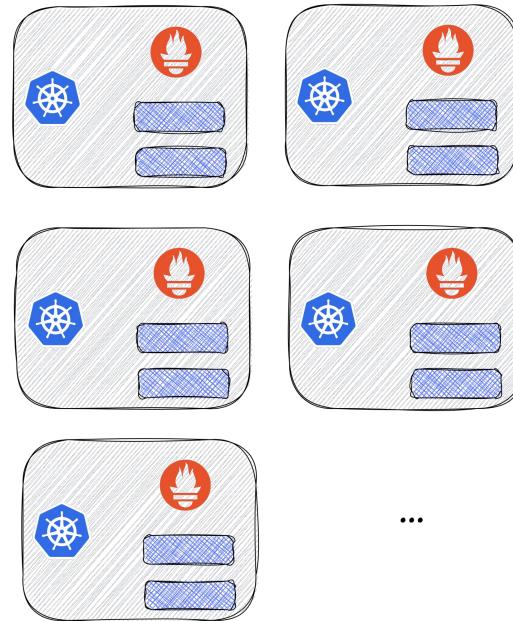


Prometheus decomposed



Limitations

- Lack of global overview of the data
- Hard to retain data for long periods of time
- Inadequate resolution for long range queries
- ...and more



Thanos: “Distributed Prometheus ++”

Thanos: Horizontally Scalable Metric Monitoring System

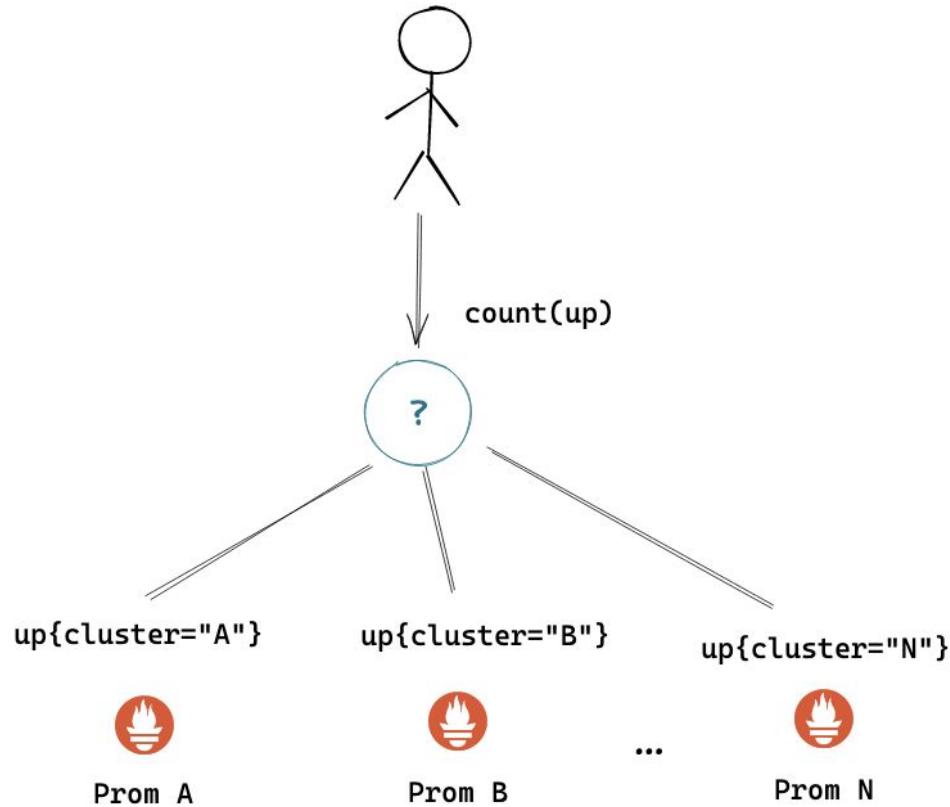
- Highly available and scalable components
- Global view
- Long term retention
- Downsampling
- Multi-Tenancy



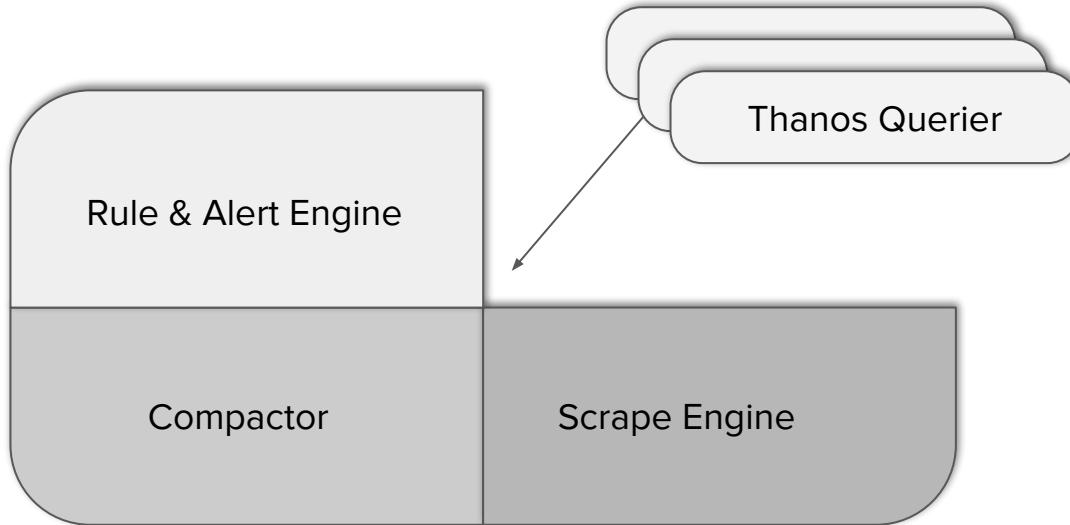
Challenge #1:

Global Query View

Global Query View



Thanos Query



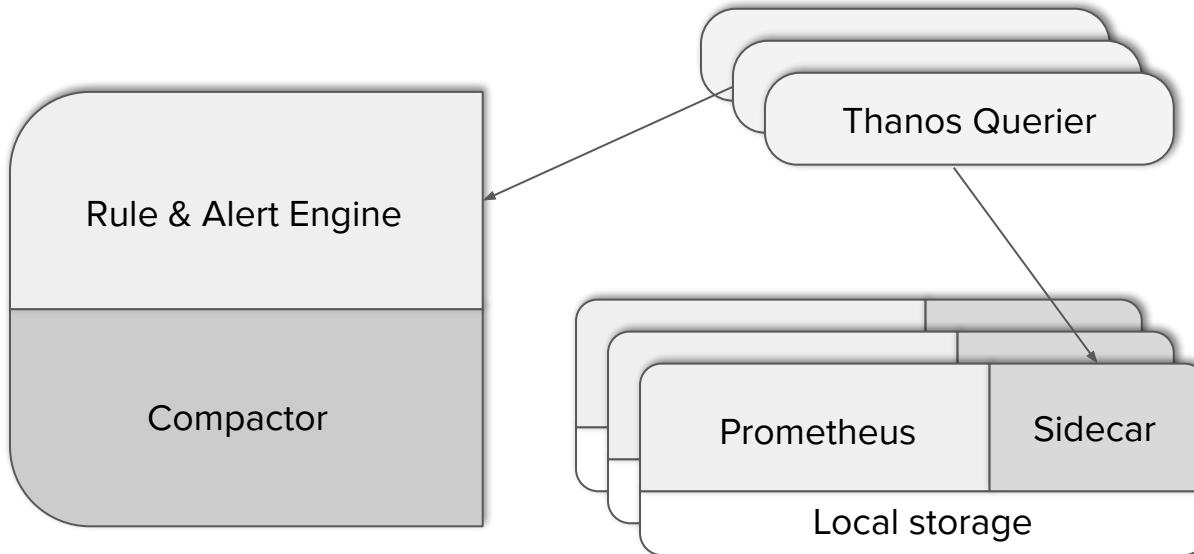
Thanos StoreAPI

```
service Store {  
    rpc Info(InfoRequest) returns (InfoResponse);  
    rpc Series(SeriesRequest) returns (stream SeriesResponse);  
    rpc LabelNames(LabelNamesRequest) returns (LabelNamesResponse);  
    rpc LabelValues(LabelValuesRequest) returns (LabelValuesResponse);  
}
```

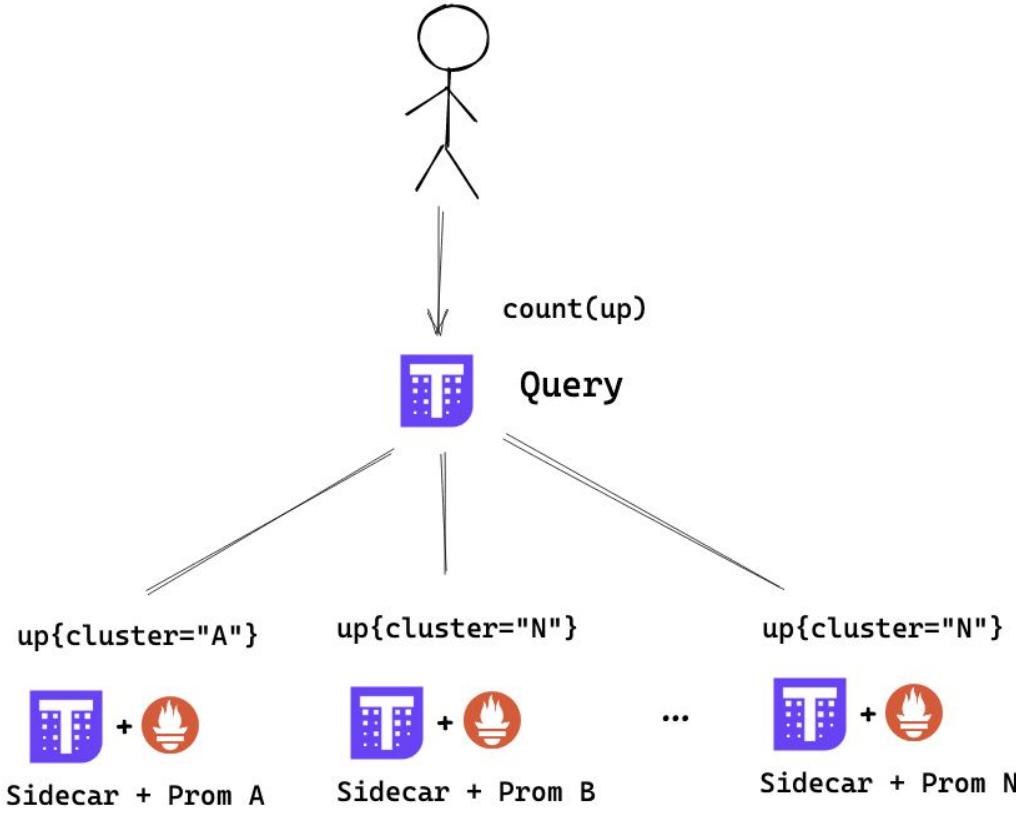
From: [rpc.proto](#)



Thanos Sidecar



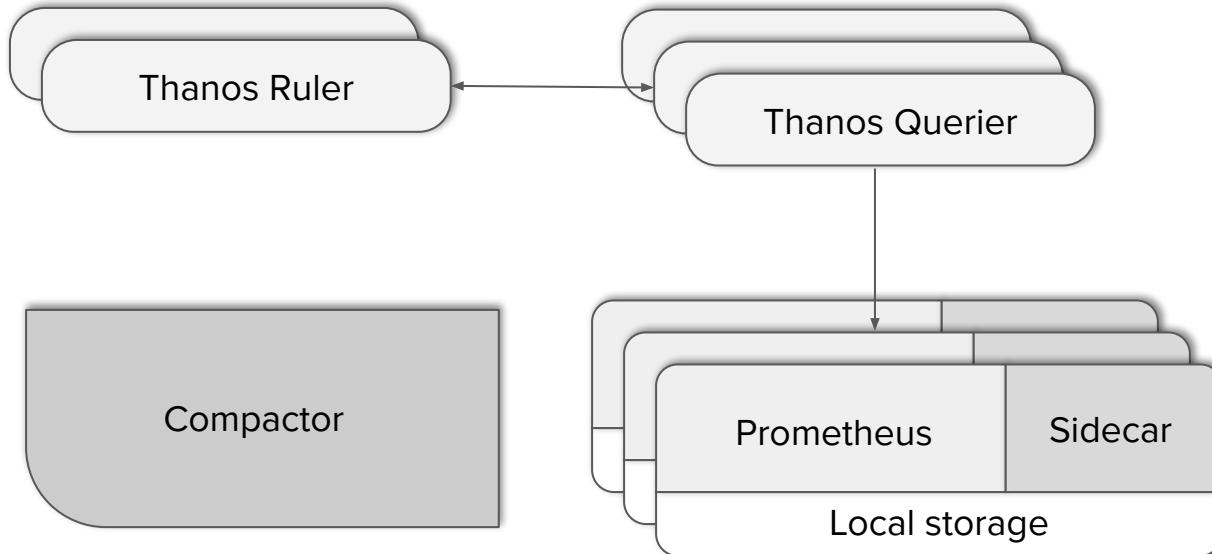
Global Query View



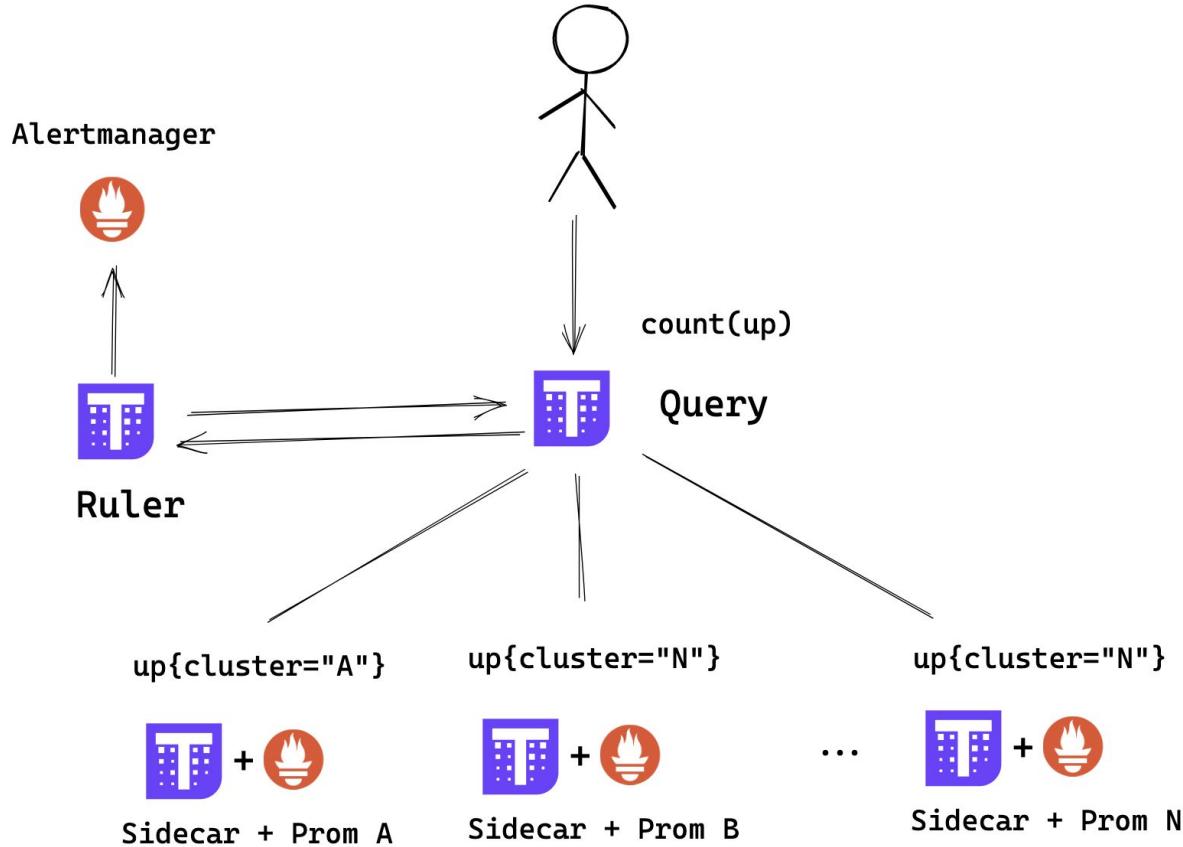
Challenge #2:

Global Rules & Alerting

Thanos Ruler



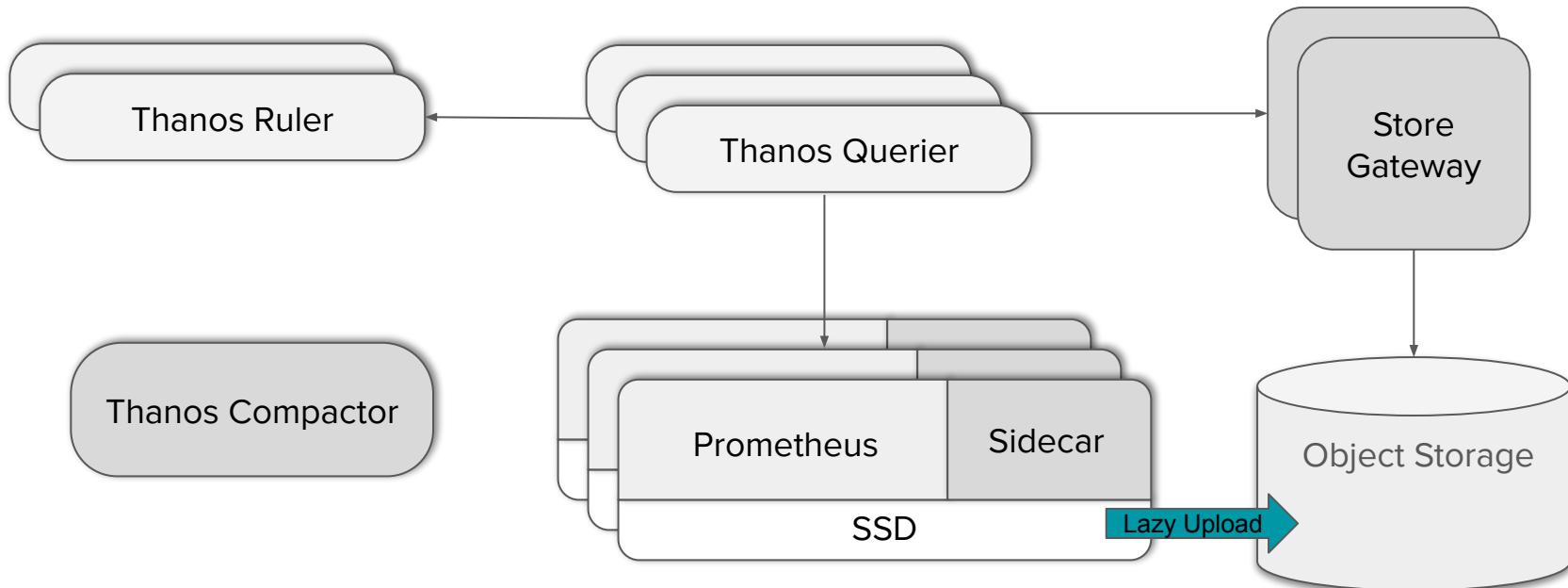
Global Rule Evaluation



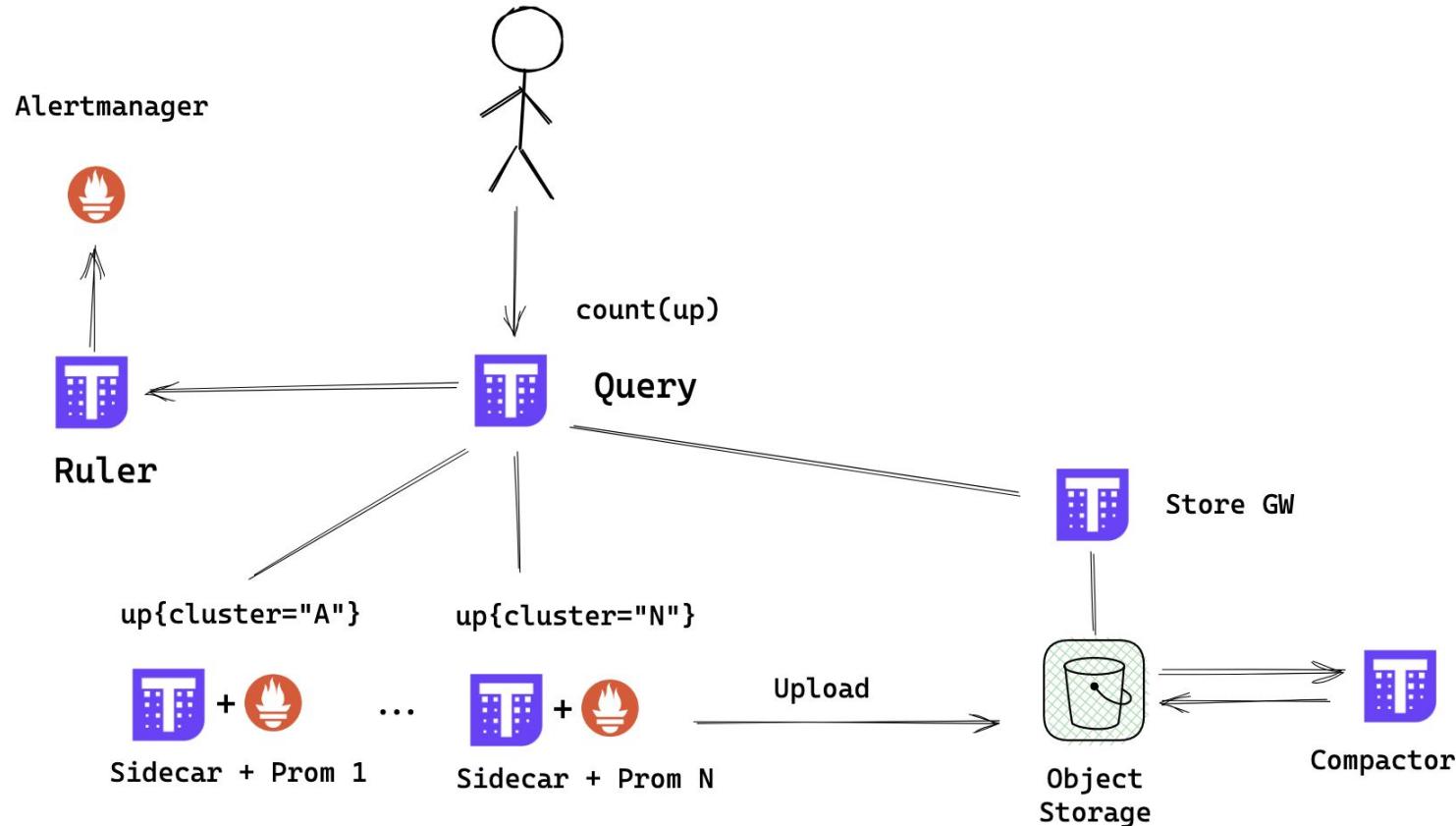
Challenge #3:

Long-term Retention

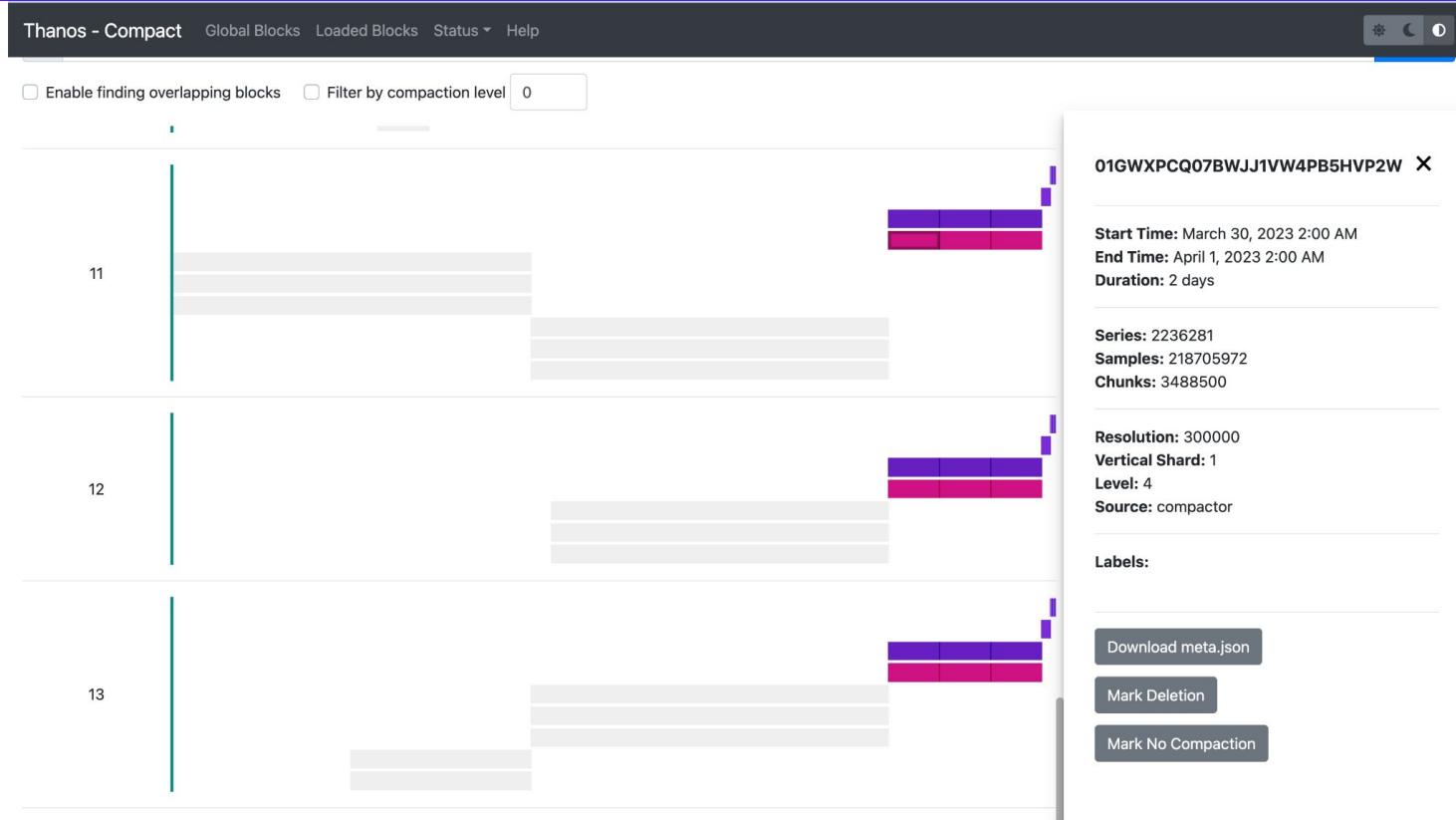
Thanos Store



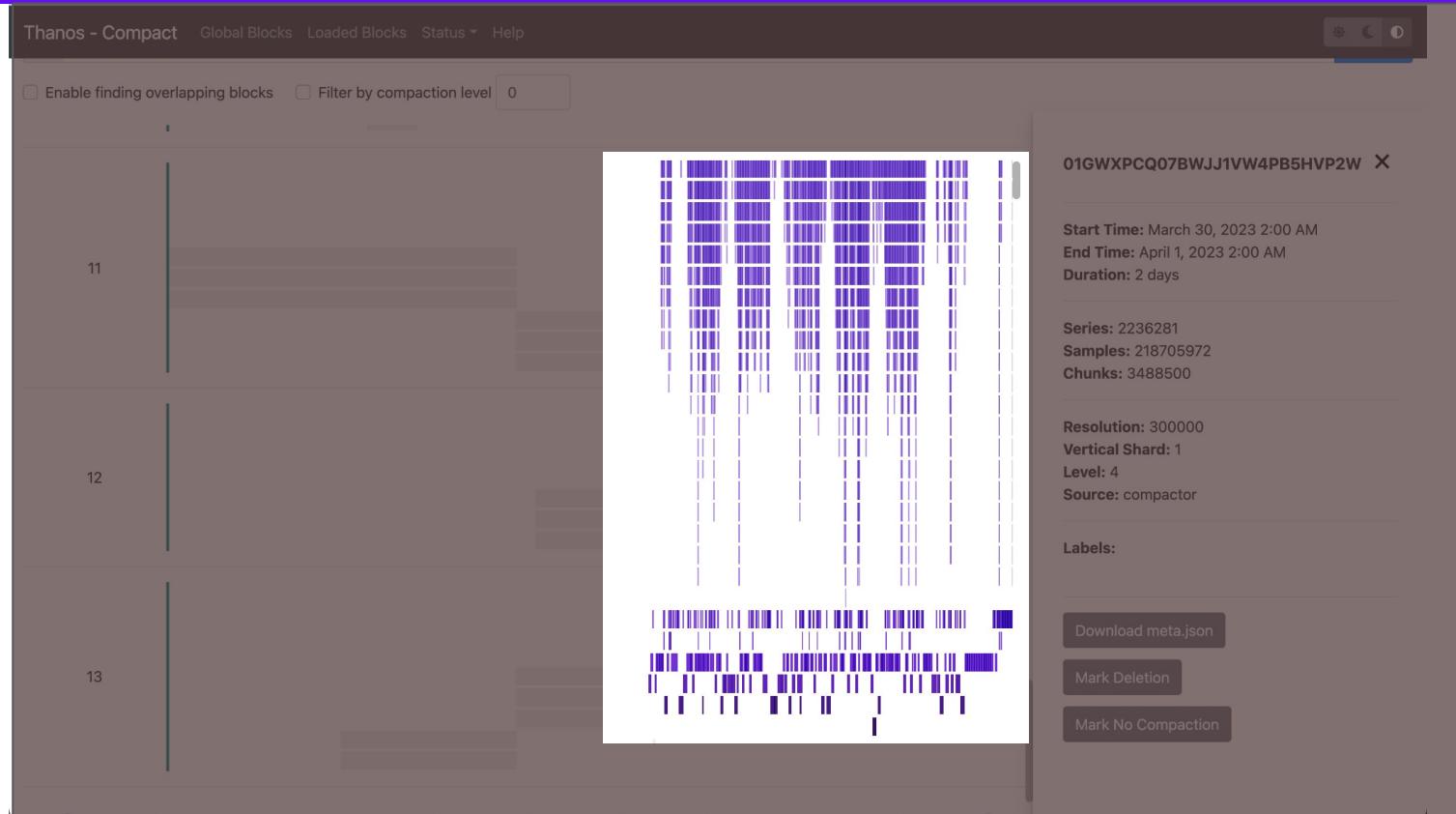
Query with Unlimited Retention



Compactor and Store UI



Compactor and Store UI



Challenge #4:

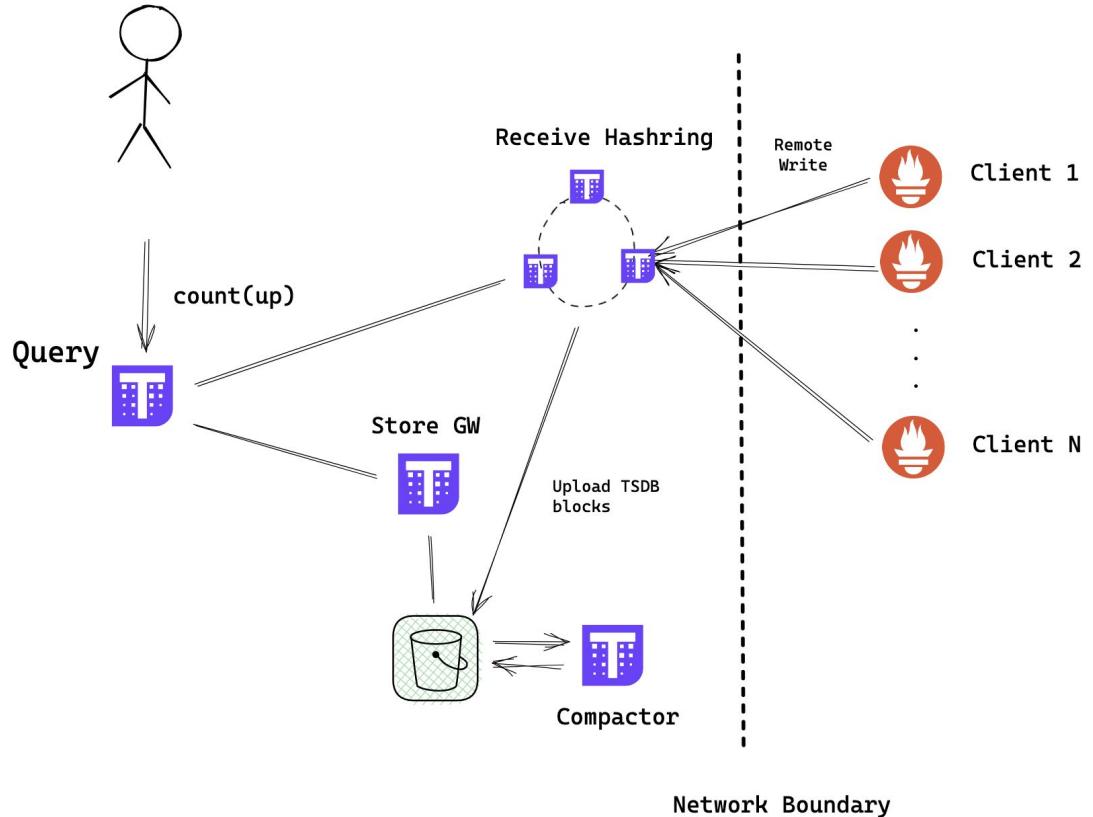
Egress-only Environments

Sidecar Architecture limitations!

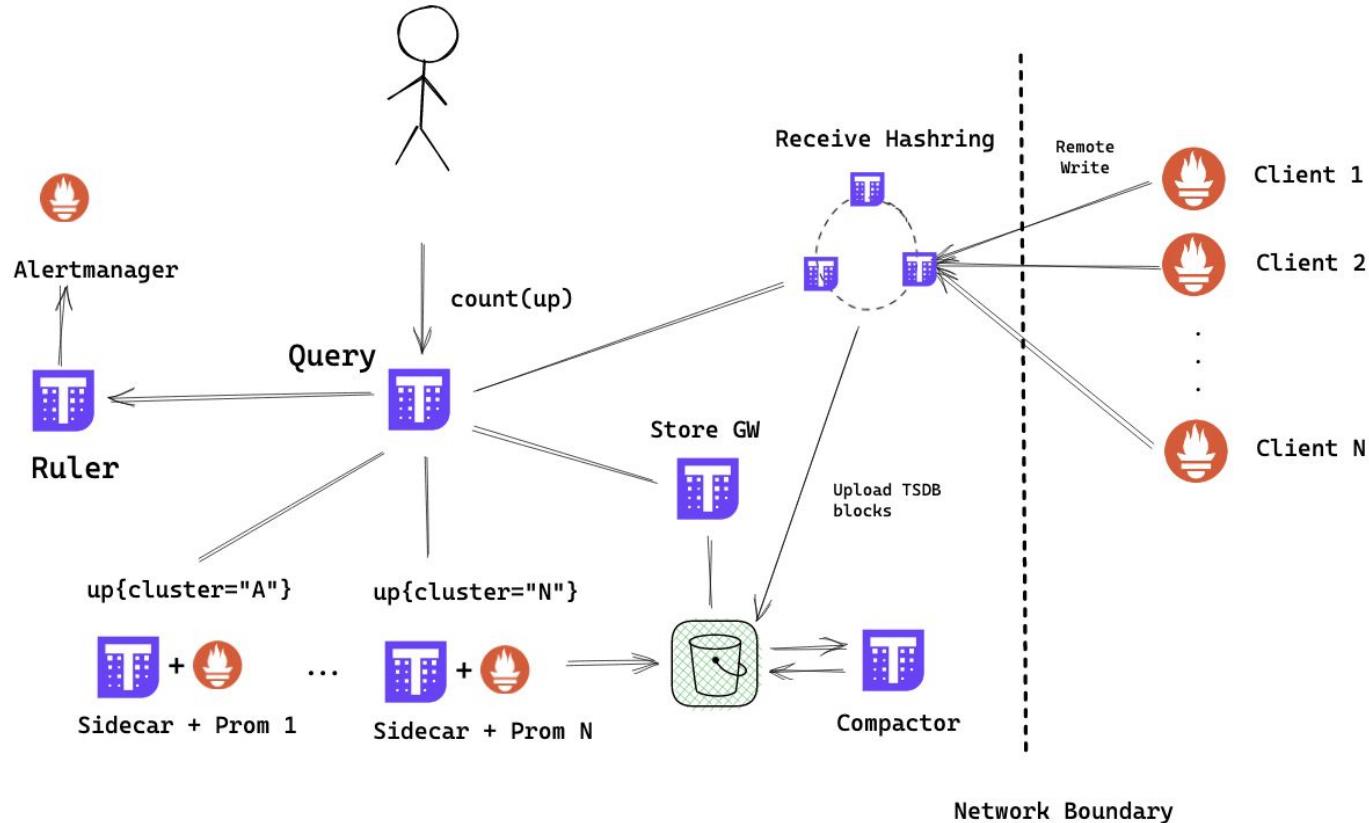
The Sidecar Architecture pattern has certain downsides:

- Needs an exposed ingress for the Querier connection
- Each sidecar needs Object Storage permissions
- Hard to design monitoring-as-a-service
- Not viable in resource constrained environments

Thanos receiver



Complex Thanos deployment

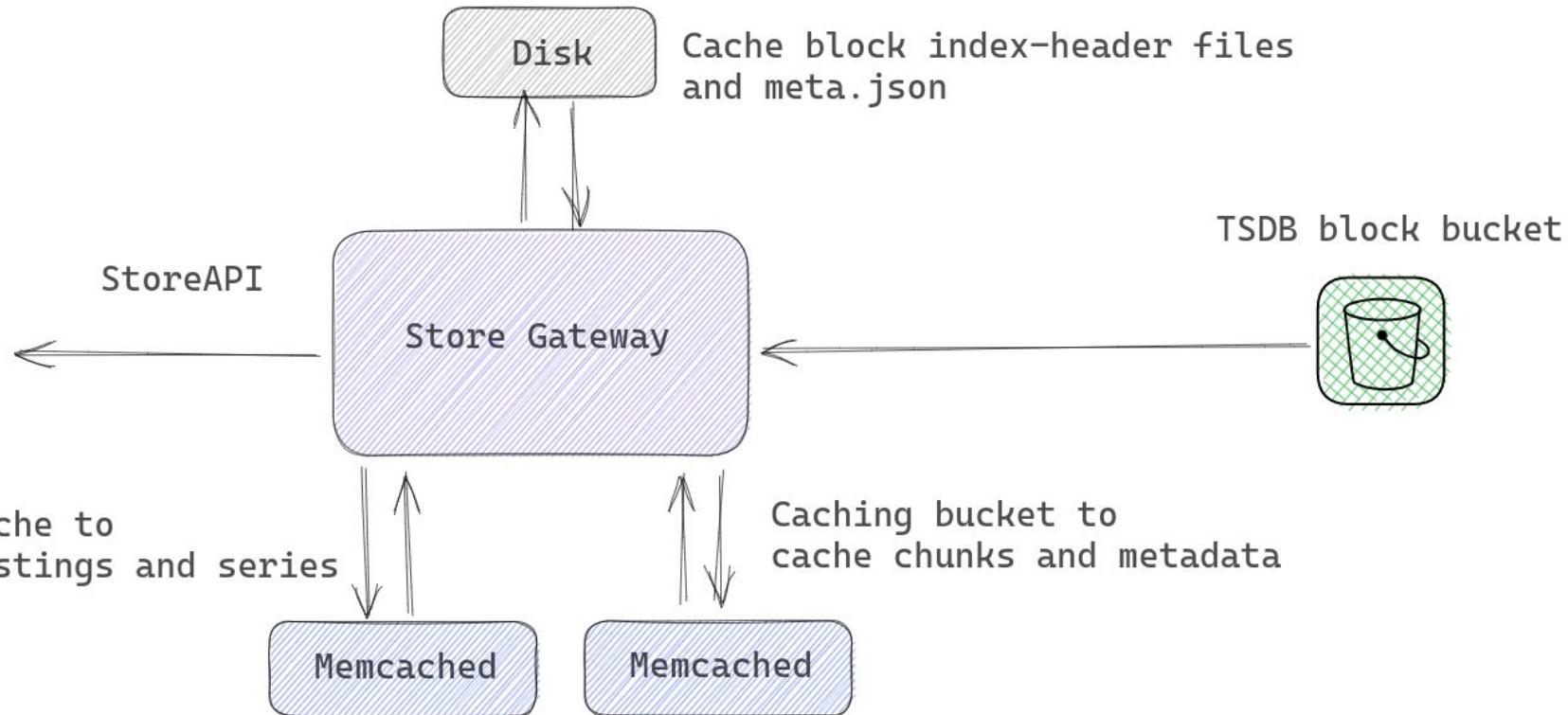


Improvements, Features & Optimizations

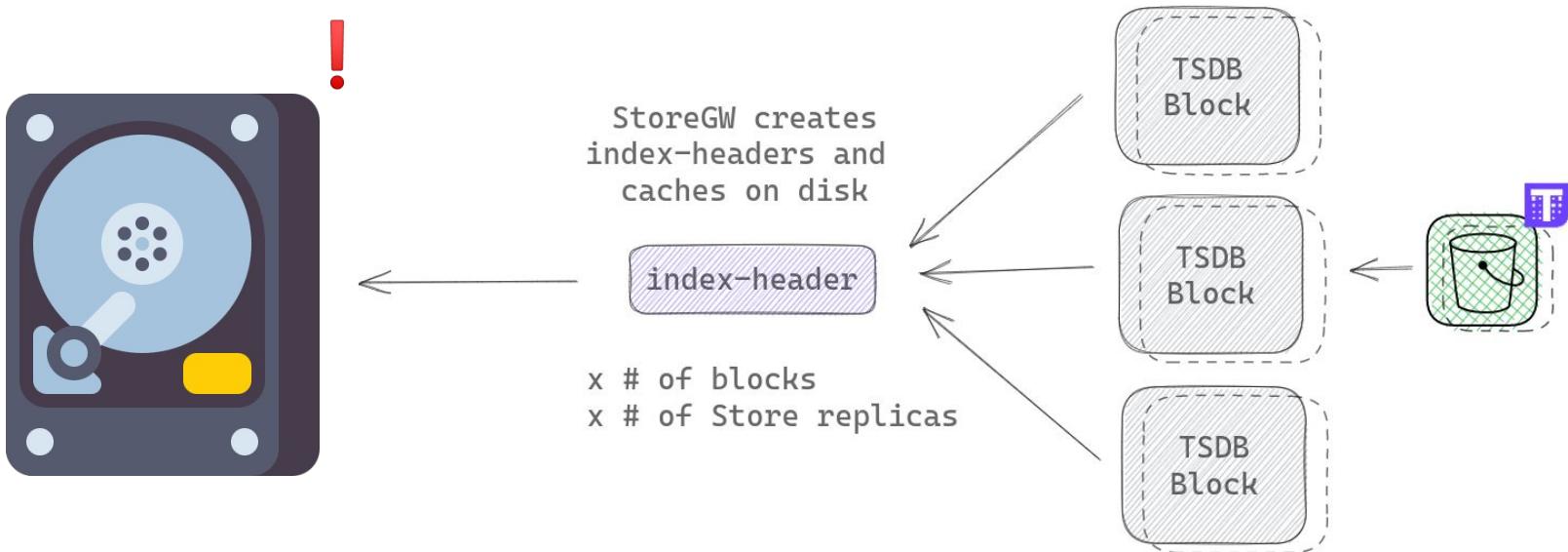
Feature Highlight #1:

Diskless Store GW!

Store Gateway



Too many index-headers!



Diskless Store Gateway

Store: Support disable cache index header file. #5773

Merged yeya24 merged 10 commits into thanos-io:main from wanjunlei:store on Jan 28

Conversation 45 · Commits 10 · Checks 14 · Files changed 8 · +546 -171

wanjunlei commented on Oct 8, 2022 • edited

Signed-off-by: wanjunlei wanjunlei@kubesphere.io

I added CHANGELOG entry for this change.
 Change is not relevant to the end user.

Changes

The block index header files can speed up store startup, but it will also take up the disk.

Suppose there is a sizeable k8s cluster, and there are many member clusters, there will be many blocks in the object storage so that the index header files will occupy more disks, and the replicas of the store need to be increased to improve the performance of the query, which will be doubled of increased disk consumption.

With this pr, the user can disable caching the index header files by `--disable-caching-index-header-file`, the store will not load index header files from the disk, also will not write them to the disk.

Verification

pull-request-size bot added the size/XXL label on Oct 8, 2022

Reviewers: yeya24, fpetkovski, matej-g

Assignees: No one—assign yourself

Labels: size/XXL

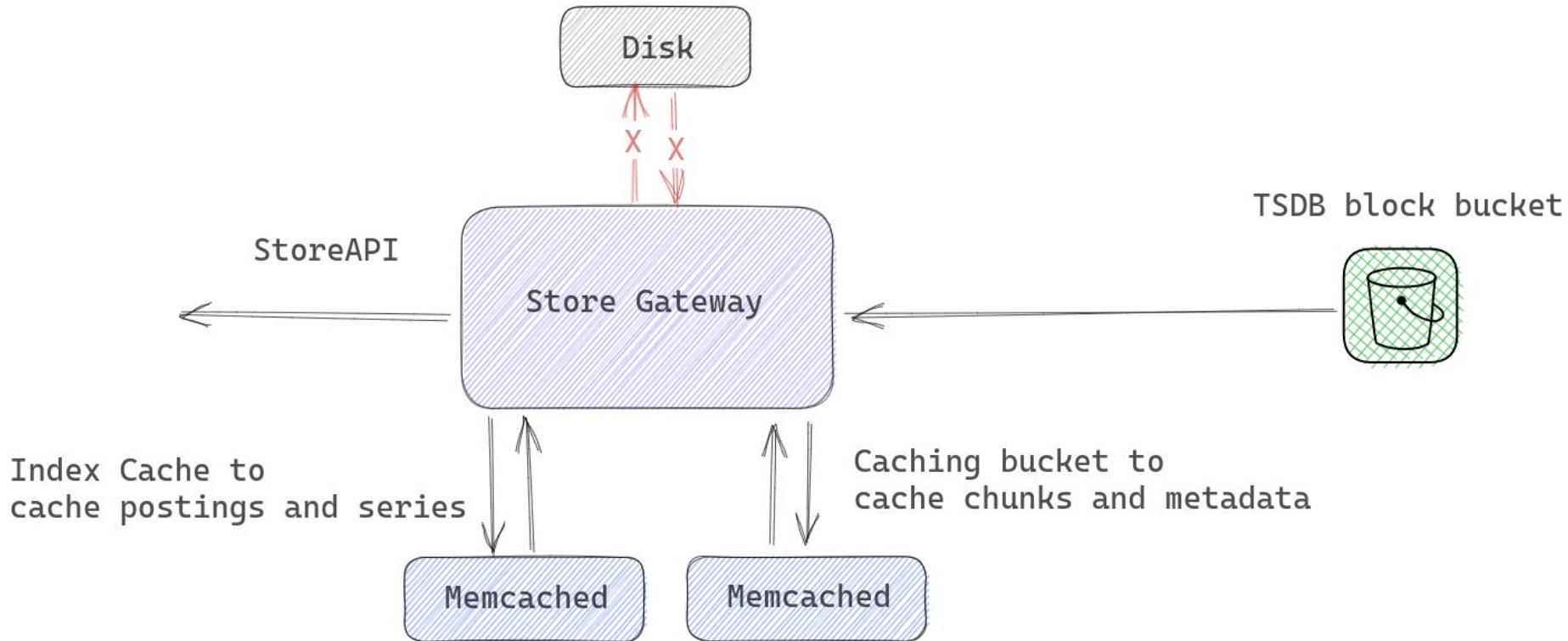
Projects: None yet

Milestone: No milestone

Development: Successfully merging this pull request may close these issues.

None yet

Diskless Store Gateway



Feature Highlight #2:

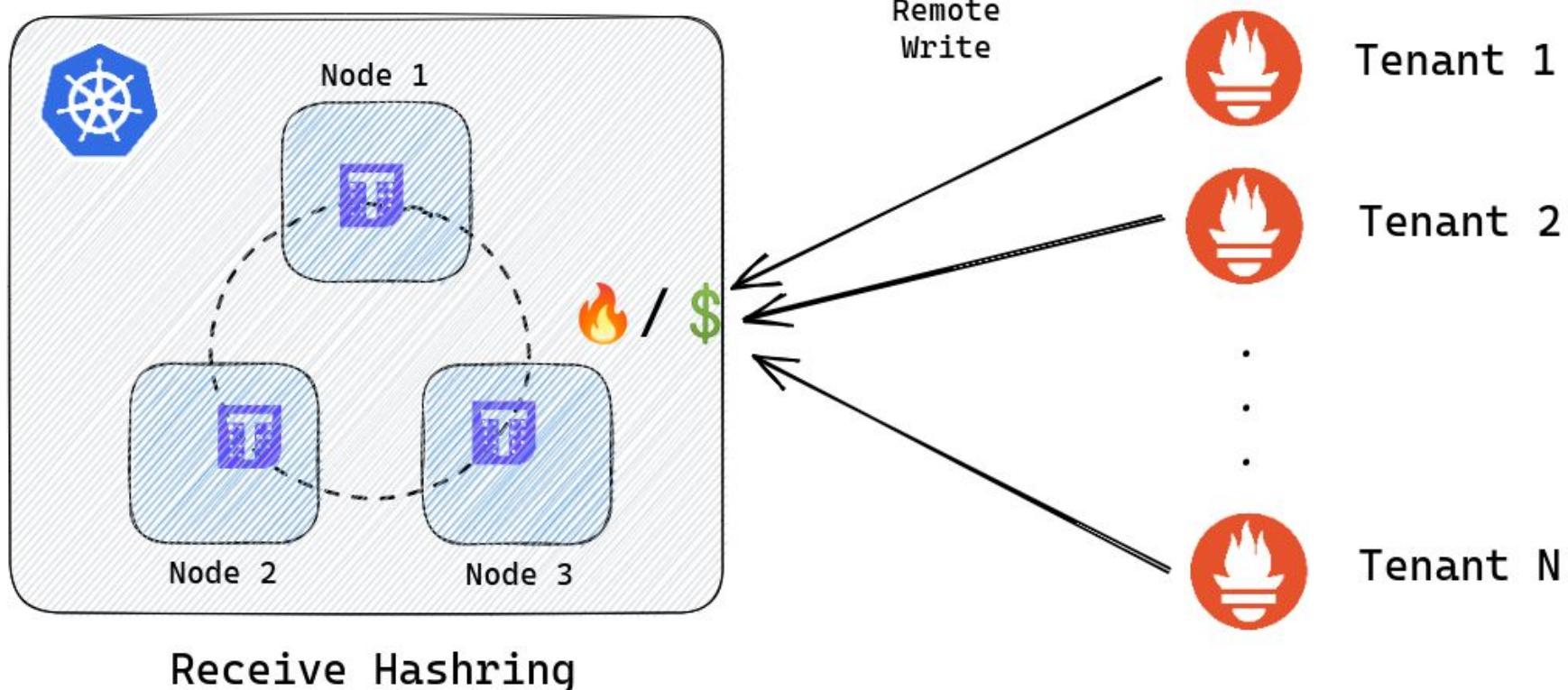
QoS Improvements with Limits!

Quality Of Service

Quality of service is the ability to provide different priorities to different applications, users, or data flows, or to guarantee a certain level of performance to a data flow.



Write Path Scale



Write Path Limits

Receive: add per request limits for remote write #5527

Merged bwplotka merged 27 commits into thanos-io:main from douglascamata:douglascamata/add-per-request-write-limits

Conversation 41 Commits 27 Checks 14 Files changed 7



douglascamata commented on Jul 20, 2022 • edited

Contributor ...

Signed-off-by: Douglas Camata 159076+douglascamata@users.noreply.github.com

- I added CHANGELOG entry for this change.
- Change is not relevant to the end user.

Changes

Implement write requests size limits in Thanos Receive.

These limits are applied per request and can be configured with the following command line arguments:

- `--receive.write-request-limits.max-size-bytes`: the maximum body size.
- `--receive.write-request-limits.max-series`: the maximum amount of series in a single remote write request.
- `--receive.write-request-limits.max-samples`: the maximum amount of samples in a single remote write request (summed from all series).

Any request above these limits will cause an 413 HTTP response (*Entity Too Large*) and should not be retried without modifications. It's up to remote write clients to split up the data and retry or completely drop it.

Another limit that was added to control concurrency on the remote write requests:

- `--receive.write-request-limits.max-concurrency`: the maximum amount remote write requests that will be concurrently processed while others wait.

Meta-monitoring based active series limiting #5520

Merged bwplotka merged 15 commits into thanos-io:main from saswatamcode:poc-meta

Conversation 118 Commits 15 Checks 14 Files changed 9



saswatamcode commented on Jul 19, 2022 • edited

Member ...

This PR adds optional meta-monitoring based active series limiting for tenants using Receive Router. As discussed in proposal #5415. 😊

- I added CHANGELOG entry for this change.
- Change is not relevant to the end user.

Changes

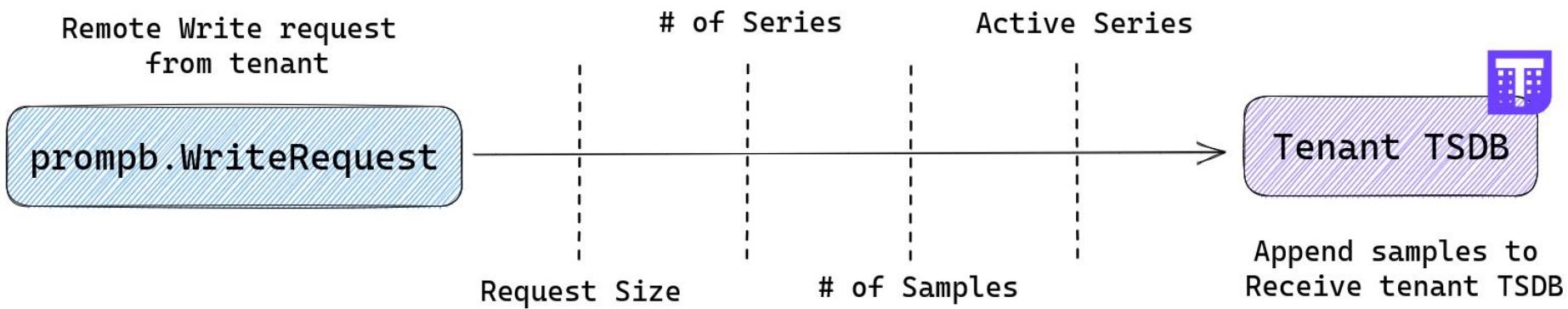
- Added four new flags to Receive,
 - `receive.tenant-limits.max-head-series` : Active series limit to set for all tenants.
 - `receive.tenant-limits.meta-monitoring-url` : Prometheus Query API compatible meta-monitoring solution.
 - `receive.tenant-limits.meta-monitoring-query` : Query to execute against meta-monitoring to get active series data. Optional.
 - `receive.tenant-limits.meta-monitoring-client` : HTTP Client config file for connecting to meta-monitoring
- Added `QueryMetamonitoring` method to Receive handler which makes Prometheus Query to meta-monitoring endpoint every 15s and caches result
- Added `isUnderLimit` method to Receive handler, refers to cache and limits remote write requests in Router/RouterIngestor mode
- Five new receive e2e tests to try out such limiting with Avalanche

Verification

Locally + e2e tests.



Write Path Limits

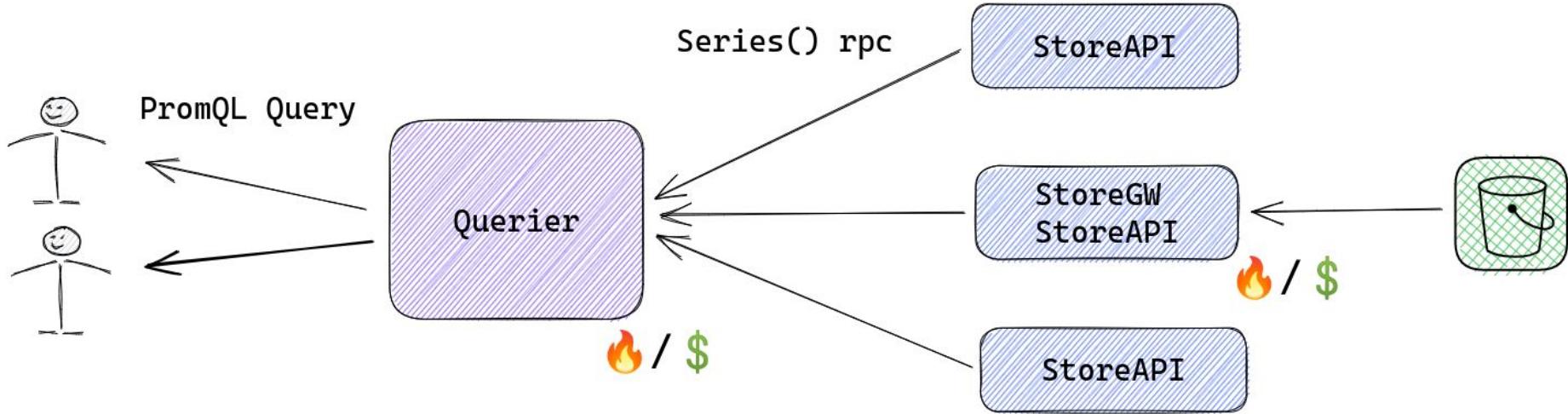


Write Path Limits

```
1 write:
2   global:
3     max_concurrency: 30
4     meta_monitoring_url: "http://localhost:9090"
5     meta_monitoring_limit_query: "sum(prometheus_tsdb_head_series) by (tenant)"
6   default:
7     request:
8       size_bytes_limit: 1024
9       series_limit: 1000
10      samples_limit: 10
11      head_series_limit: 1000
12    tenants:
13      acme:
14        request:
15          size_bytes_limit: 0
16          series_limit: 0
17          samples_limit: 0
18          head_series_limit: 2000
19      ajax:
20        request:
21          series_limit: 50000
22          samples_limit: 500
```



Query Path Scale



Query Path Limits

Instrumented servers #6074

Merged matej-g merged 12 commits into `thanos-io/main` from `fpetkovski:instrumented-servers` on Feb 13

[Conversation](#) 32 · [Commits](#) 12 · [Checks](#) 14 · [Files changed](#) 14

 fpetkovski commented on Jan 26 • edited

This PR adds an InstrumentedStoreServer that exposes metrics about Series requests and uses it as a decorator around all Store APIs. The instrumented store currently exposes two histogram metrics, series requested and chunks requested. Additional metrics can be added as needed.

This PR also implements a RateLimitedStoreServer which can be used to apply various limits to Series calls in components that implement the Store API.

Rate limits are disabled by default but can be enabled selectively for each individual Thanos component.

I added CHANGELOG entry for this change.
 Change is not relevant to the end user.

Changes

- Expose histogram metrics from stores about series and chunks per Series request.
- Allow configuring limits for series and chunks per Series request.

Verification

Manual verification and tests.

  3

store: add downloaded bytes limit #5801

Merged GiedriusS merged 8 commits into `thanos-io/main` from `GiedriusS:add_downloaded_bytes_limit` on Oct 27, 2022

[Conversation](#) 13 · [Commits](#) 8 · [Checks](#) 14 · [Files changed](#) 9

 GiedriusS commented on Oct 20, 2022 • edited

Changes

Adds the ability to limit downloaded bytes in Thanos Store for LabelNames/LabelValues/Series. This is needed to make capacity planning easier and because current limits aren't that effective since to calculate series/chunks limit, we first need to download all matching postings.

Closes [#5733](#).

I added CHANGELOG entry for this change.
 Change is not relevant to the end user.

Verification

E2E tests.

 2

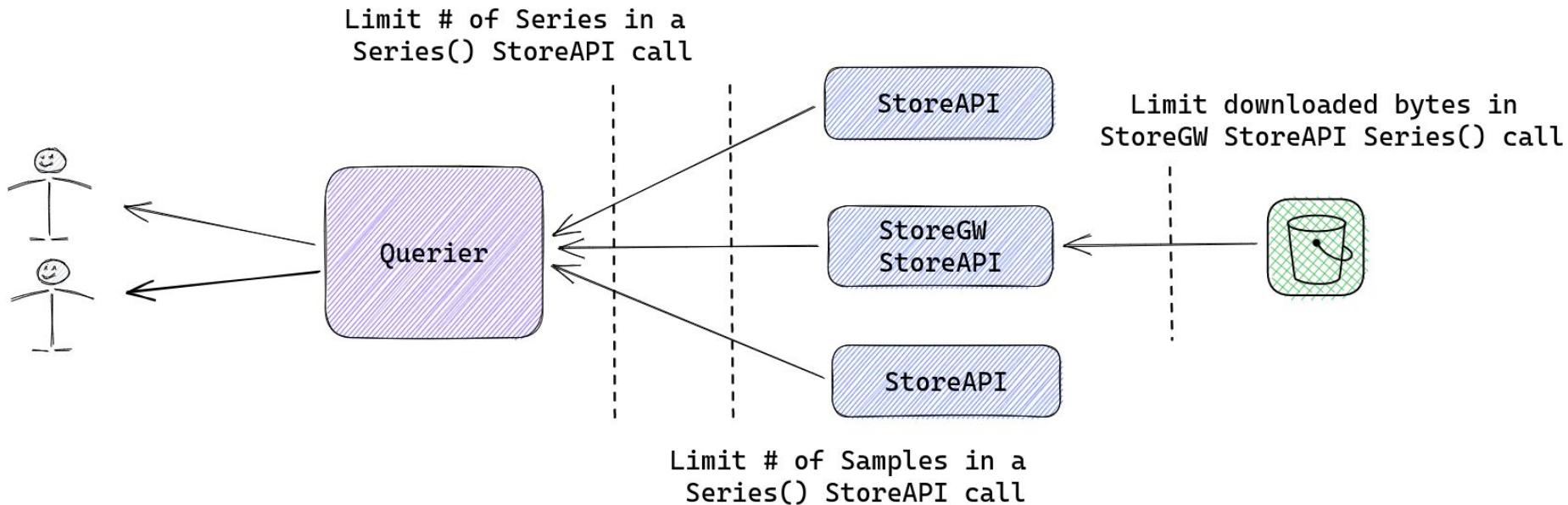
 GiedriusS added 3 commits 6 months ago

 store: add downloaded bytes limit ... d8fbba06

 store: add bytesLimiter to LabelNames, LabelValues ... 2b3330c



Query Path Limits



Feature Highlight #3:

Consistent Hashing in Receive

Ketama Hashring

Consistent hashing #5408

Merged bwplotka merged 3 commits into `thanos-io:main` from `fpetkovski:consistent-hashing` on Jun 16, 2022

Conversation 24 Commits 3 Checks 14 Files changed 8

 fpetkovski commented on Jun 8, 2022 • edited Member ...
This commit adds support for distributing series in Receivers using consistent hashing based on the libketama algorithm.
The following article provides a great summary of all available options: <https://dgryski.medium.com/consistent-hashing-algorithmic-tradeoffs-ef6b8e2fcae8>.
We can also use Multi-Probe Consistent Hashing which might have a better performance.
Fixes #4972
 I added CHANGELOG entry for this change.
 Change is not relevant to the end user.

Changes

Add support for consistent hashrings in Thanos Receiver.

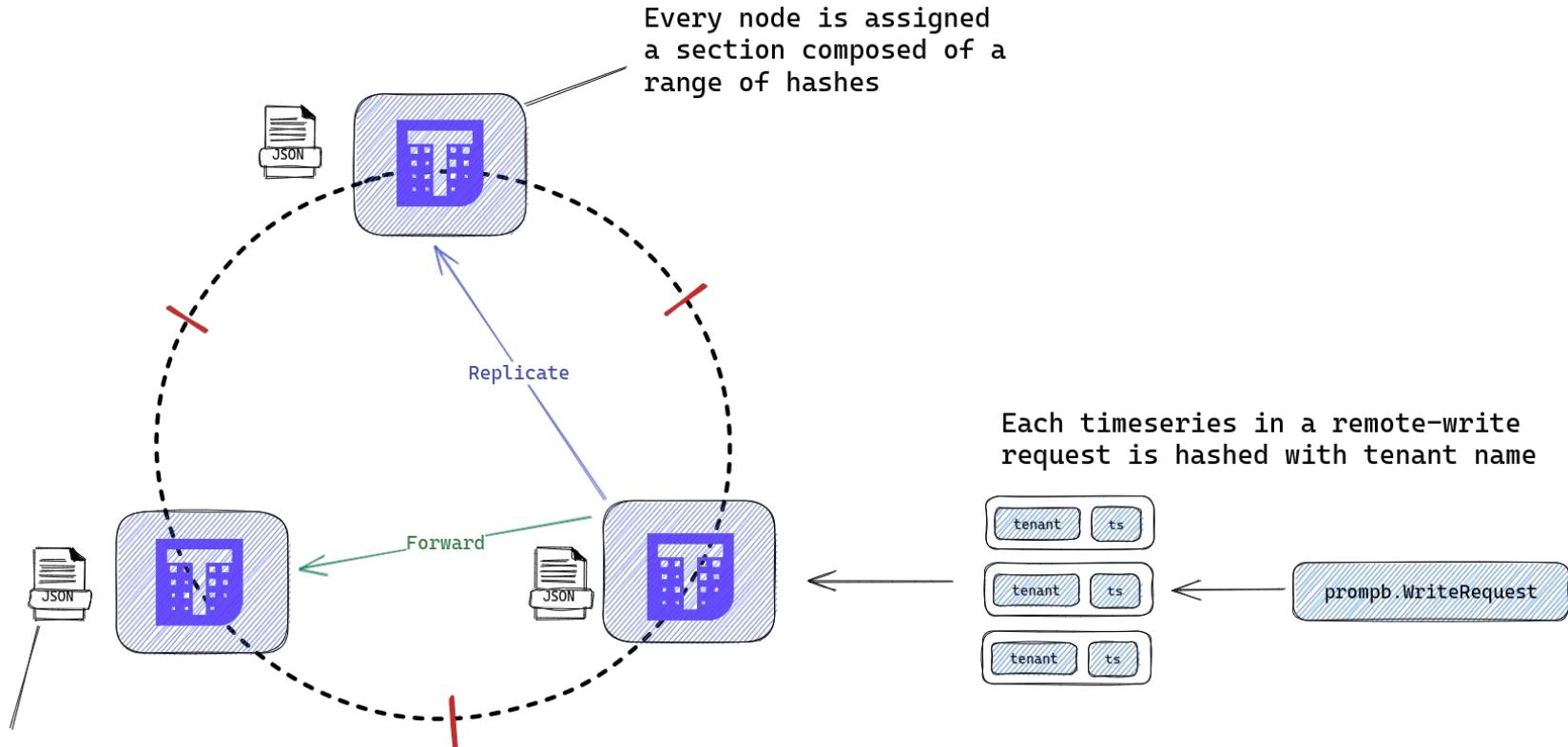
Verification

Unit tests.

4 2

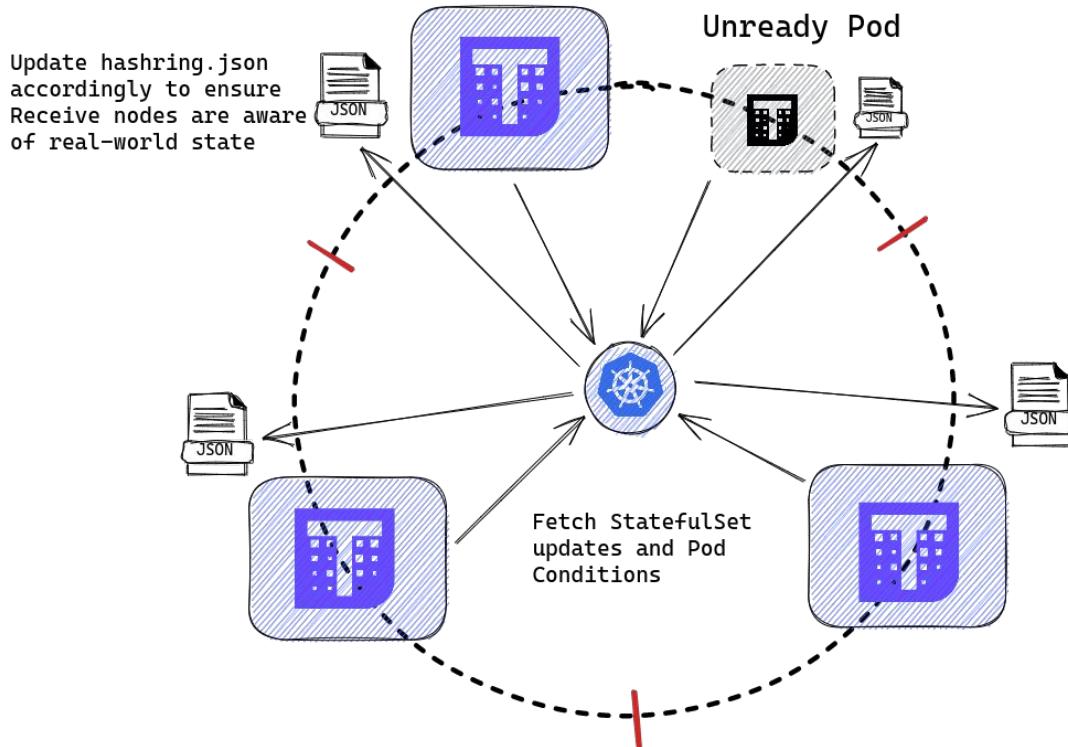


Ketama Hashring



Thanos Receive Controller

<https://github.com/observatorium/thanos-receive-controller/>

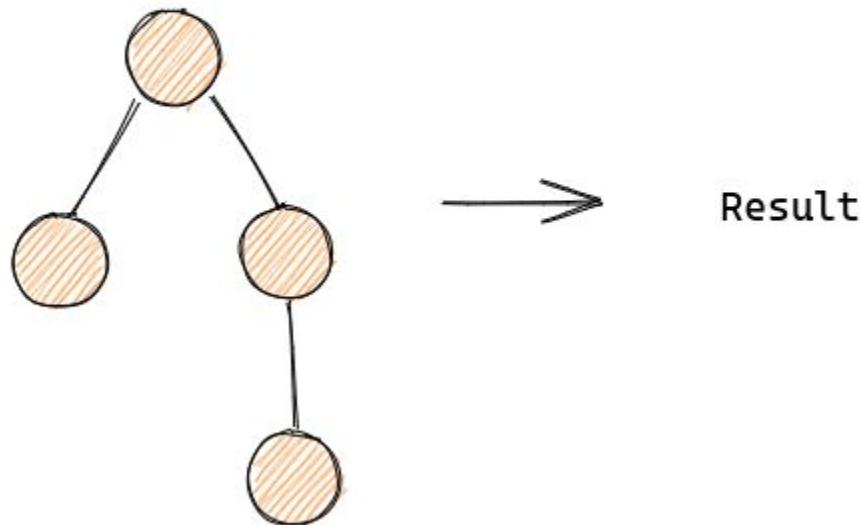


Feature Highlight #4:

New PromQL Engine!

Prometheus PromQL Engine

Query Parsed into AST
Evaluate as we parse
and return value, all in a single thread



Thanos PromQL Engine: Design

<https://paperhub.s3.amazonaws.com/dace52a42c07f7f8348b08dc2b186061.pdf>

120 IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 6, NO. 1, FEBRUARY 1994

Volcano—An Extensible and Parallel Query Evaluation System

Goetz Graefe

Abstract—To investigate the interactions of extensibility and parallelism in database query processing, we have developed a new dataflow query execution system called Volcano. This paper describes the Volcano effort and provides a rich environment for research in parallelization in database systems design.

Volcano uses a standard interface for operators, allowing easy addition of new operators and their implementations. Operations on individual data items or item sets are handled by the query processing system via support functions. The semantics of support functions is prescribed; any data type and any complex object an operation can be realized.

Volcano is extensible with respect to both specific metrics and operators, algorithms, data models, and data types.

Volcano includes two parallel execution models. The choice between them is controlled by the query evaluation plan, which allows delaying selected operations until e.g., for embedding query fragments into a main query. Both execution models are based on partitioned datasets and both support horizontal parallelism, translating between demand-driven data processes and data-driven dataflow between processes.

All operators, with the exception of the exchange operator, have been designed and implemented in a single environment, and parallelized using the exchange operator. Some operators not yet designed can be parallelized using the exchange operator if they use and provide the interfaces of data manipulation and data exchange. This makes Volcano the first query execution engine that is truly orthogonal, making Volcano the first truly orthogonal query execution engine that is effective.

Index Terms—Dynamic query evaluation plans, extensible database systems, iterators, operator model of parallelization, query execution.

1. INTRODUCTION

IN ORDER to investigate the interactions of extensibil-

ity such as a user-friendly query language, a system for instances (record definitions), a catalog, and so on. Because of this focus, Volcano is an experimental vehicle for a multitude of research topics, many of them open-ended, which results

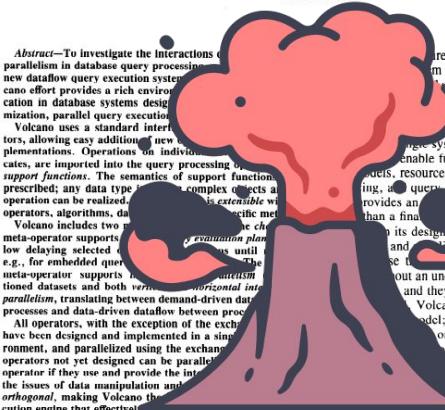
in requirements that have not been in a query system before. First, it is modular and extensible future research, e.g., on algorithms, data models, resource allocation, parallel execution, load balancing, and query optimization heuristics. Thus, Volcano provides an environment for experimental research rather than a final product in itself. Second, it is extensible in its design for different use and research.

Third, it is important for this purpose to use two parallel execution models to begin working on Volcano without an understanding of the entire design and implementation of the system, and they permit several concurrent student projects.

Volcano's design does not presume any particular data model; the only assumption is that query operators are used for transforming sets of items using some kind of operators. To achieve data model independence, Volcano consistently separates set processing from item processing, as is standard and inherent in the Volcano design. Finally, the execution and manipulation of data is orthogonal to the operators, as described later).

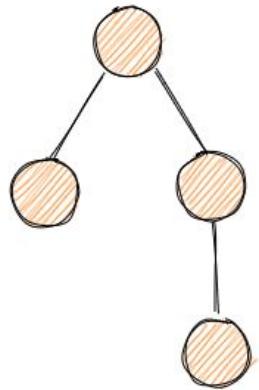
Fourth,

to free algorithm design, implementation, debugging, tuning, and initial experimentation from the intricacies of parallelism but to allow experimentation with parallel query processing. Volcano can be used as a single-process or as a parallel system. Single-process query evaluation plans can already be parallelized easily on shared-memory machines and soon also on distributed-memory machines. Fifth, Volcano is realistic in

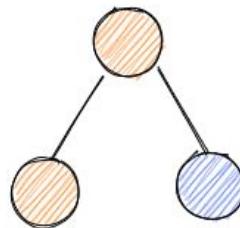


Thanos PromQL Engine: How it works?

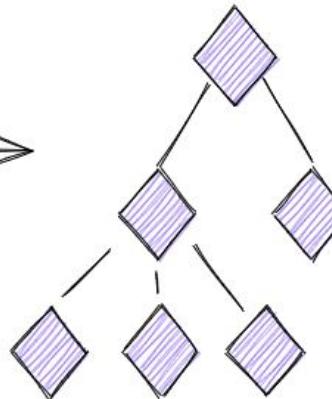
Query Parsed into AST



Logical Plan optimizes query tree
(if possible)



Physical plan converts
Query tree to Operator tree
and executes Query using multiple cores



Result

Thanos PromQL Engine: Operators

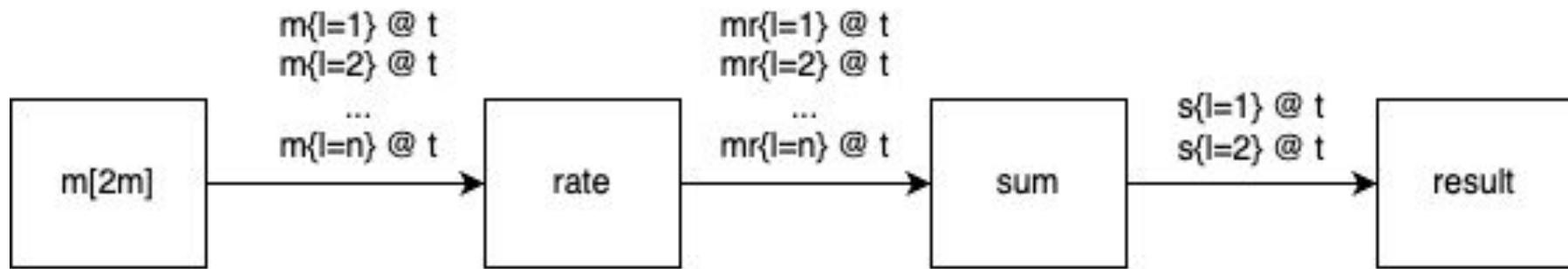
```
// VectorOperator performs operations on series in step by step fashion.
type VectorOperator interface {
    // Next yields vectors of samples from all series for one or more execution steps.
    Next(ctx context.Context) ([]StepVector, error)

    // Series returns all series that the operator will process during Next results.
    // The result can be used by upstream operators to allocate output tables and buffers
    // before starting to process samples.
    Series(ctx context.Context) ([]labels.Labels, error)

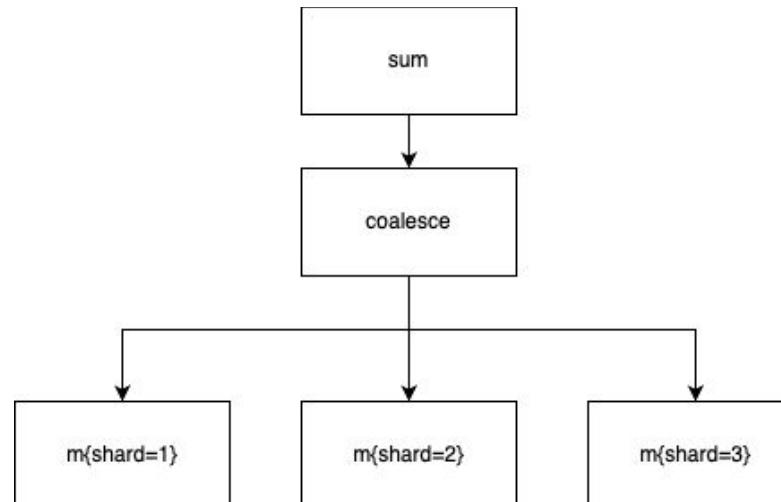
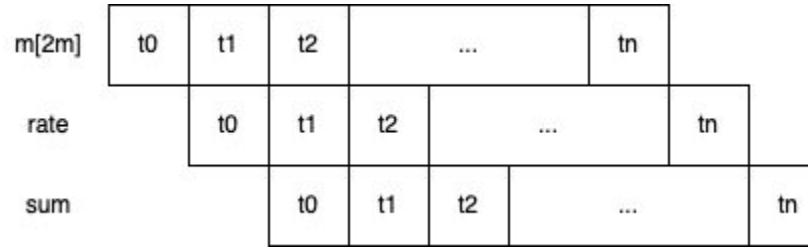
    // GetPool returns pool of vectors that can be shared across operators.
    GetPool() *VectorPool

    // Explain returns human-readable explanation of the current operator and optional nested operators.
    Explain() (me string, next []VectorOperator)
}
```

Thanos PromQL Engine: Operator flow



Thanos PromQL Engine: Parallelism

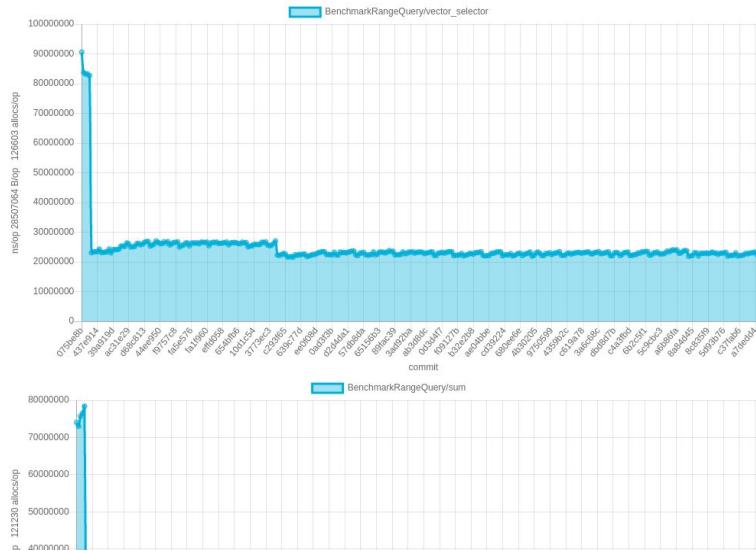


Thanos PromQL Engine: Benchmarks

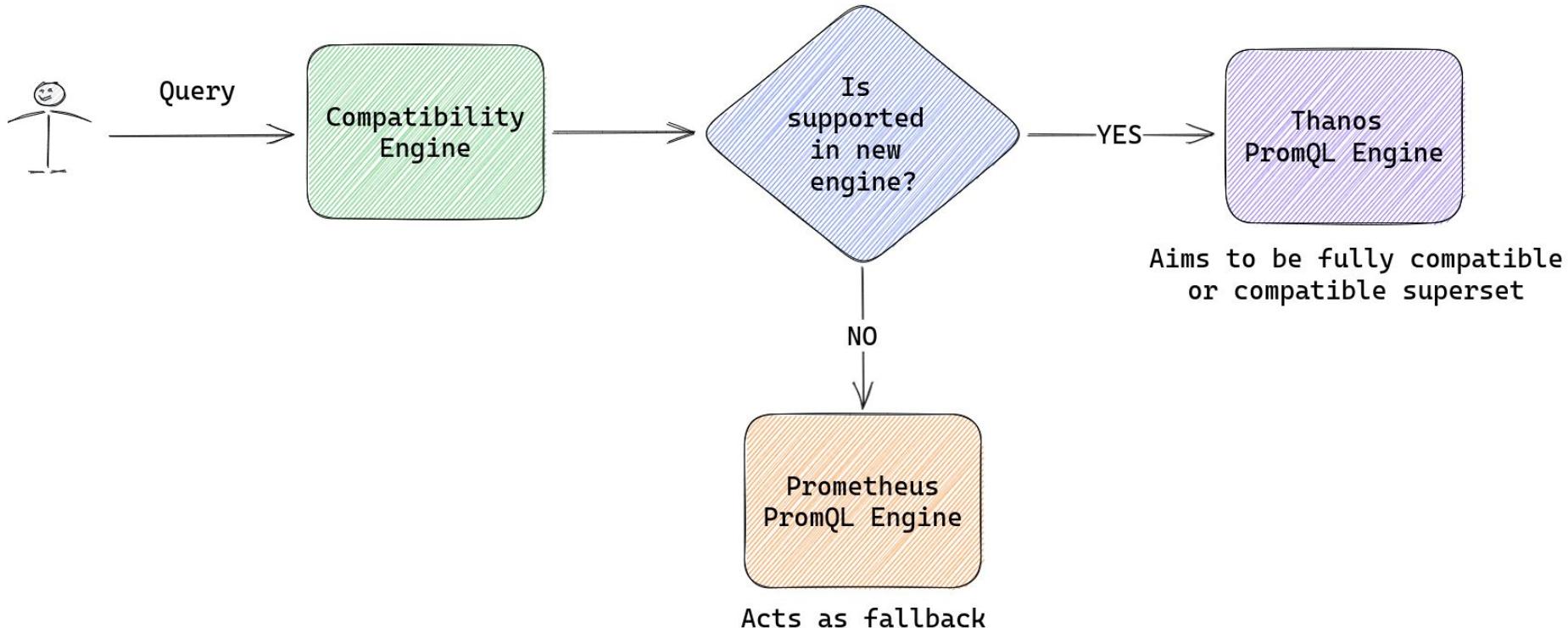
<https://thanos-community.github.io/promal-engine/dev/bench/>

Repository: <https://github.com/thanos-community/promql-engine>

Go Benchmark



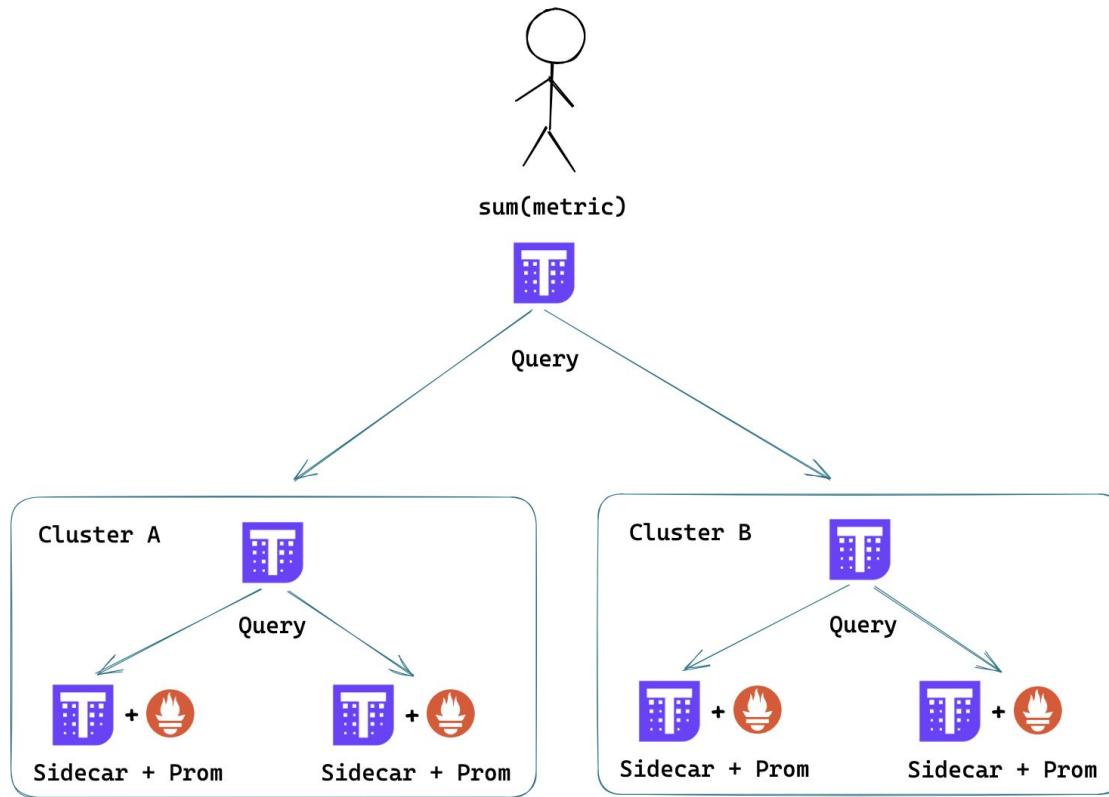
Thanos PromQL Engine: Compatibility



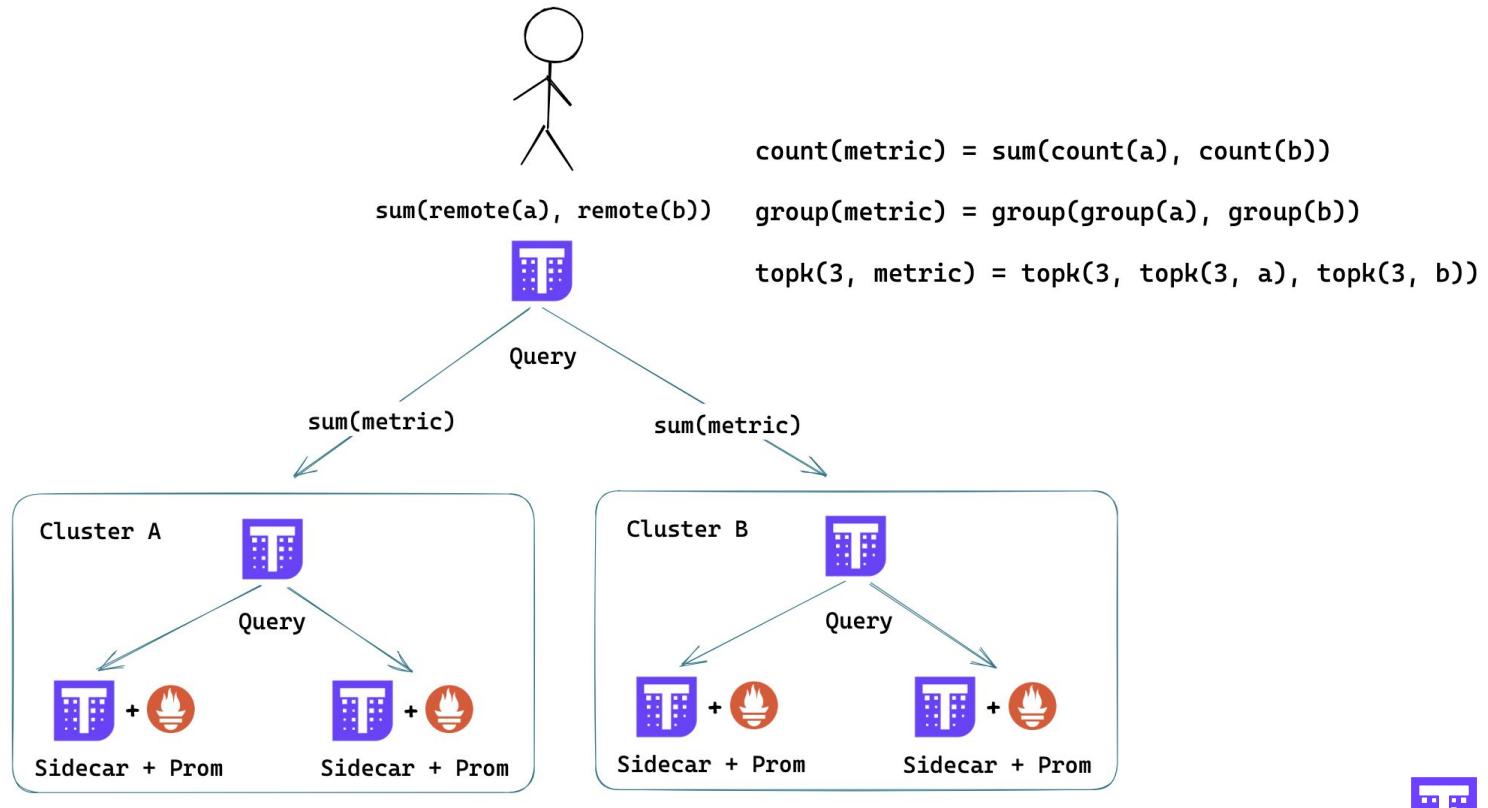
Feature Highlight #5:

Distributed Execution!

Distributed execution



Distributed execution



Native histograms

Add Native Histogram Support #5907

[Open](#) 3 of 7 tasks rabenhorst opened this issue on Nov 18, 2022 · 4 comments



rabenhorst commented on Nov 18, 2022 · edited

Is your proposal related to a problem?

Experimental native histogram support was released with Prometheus v0.40.x. [#5896](#) updated Thanos to Prometheus v0.40.1 but without implementing native histograms.

I propose the implementation of native histogram support (behind a feature flag) as a next step in follow up PR(s).

Describe the solution you'd like

We need to go through all Thanos components and check what needs to be changed for native histogram support. So far we identified the following changes are required (*list will be updated on new findings*):

- Sidecar | [sidecar, query, receiver: Native histogram support #6032](#)
 - Add native histograms to storepb and prompb types
- Query | [sidecar, query, receiver: Native histogram support #6032](#)
 - Add native histogram support for dedup and query iterators
 - Update UI for properly displaying native histograms
- Query Frontend | [query frontend, query UI: Native histogram support #6071](#)
 - Update types
- Receive | [sidecar, query, receiver: Native histogram support #6032](#)
 - Update writer to append histograms
- Compactor | w.i.p.
 - Implement native histogram downsampling
- Store | [Support native histograms in store-gateway #6056](#)

Detailed investigation still needs to be done for the following components.

- Rule

For each component we need to make sure to have tests in place after implementation and e2e test for functionality provided by multiple components (e.g. ingestion, querying, etc.).



sidecar, query, receiver: Native histogram support #6032

[Merged](#) GiedriusS merged 8 commits into [thanos-io:main](#) from [rabenhorst:native-histograms](#) [on Jan 24](#)

query frontend, query UI: Native histogram support #6071

[Open](#) rabenhorst wants to merge 2 commits into [thanos-io:main](#) from [rabenhorst:native-histograms-qfe](#) [on Jan 24](#)

Conversation 21 Commits 2 Checks 14 Files changed 23



rabenhorst commented on Jan 24 · edited

Contributor ...

As followup to [#6032](#), this PR adds native histogram support for query frontend and query UI. Additionally this PR adds native histogram support in the `ApplyCounterResetsSeriesIterator`, which is required for histogram rates.

- I added CHANGELOG entry for this change.
- Change is not relevant to the end user.

Changes

- Native histogram support for query frontend
- Native histogram support for query UI (table and graph)
- Native histogram support in `ApplyCounterResetsSeriesIterator`
- [Enable native histograms rates](#).

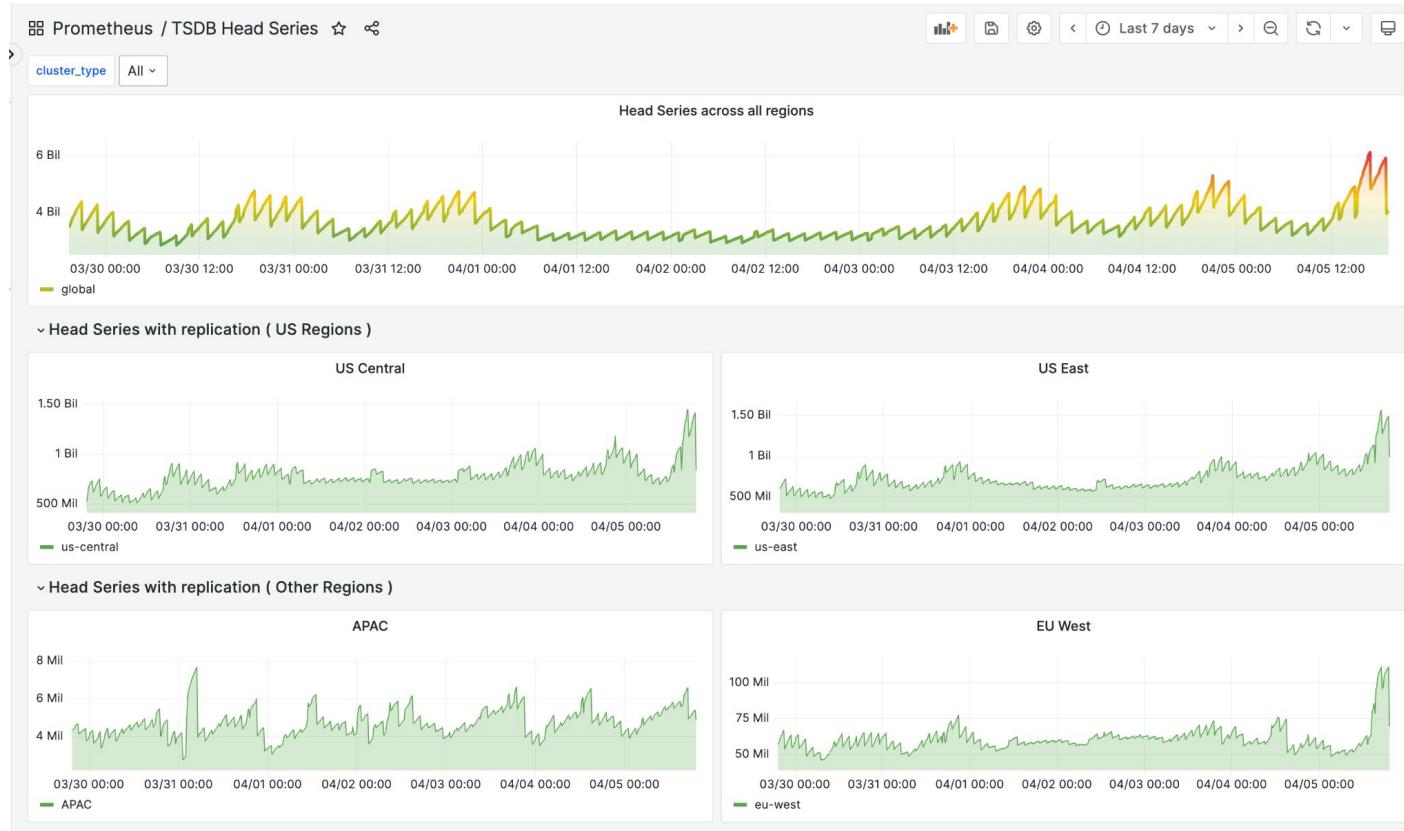
Verification

Unit tests and e2e tests.



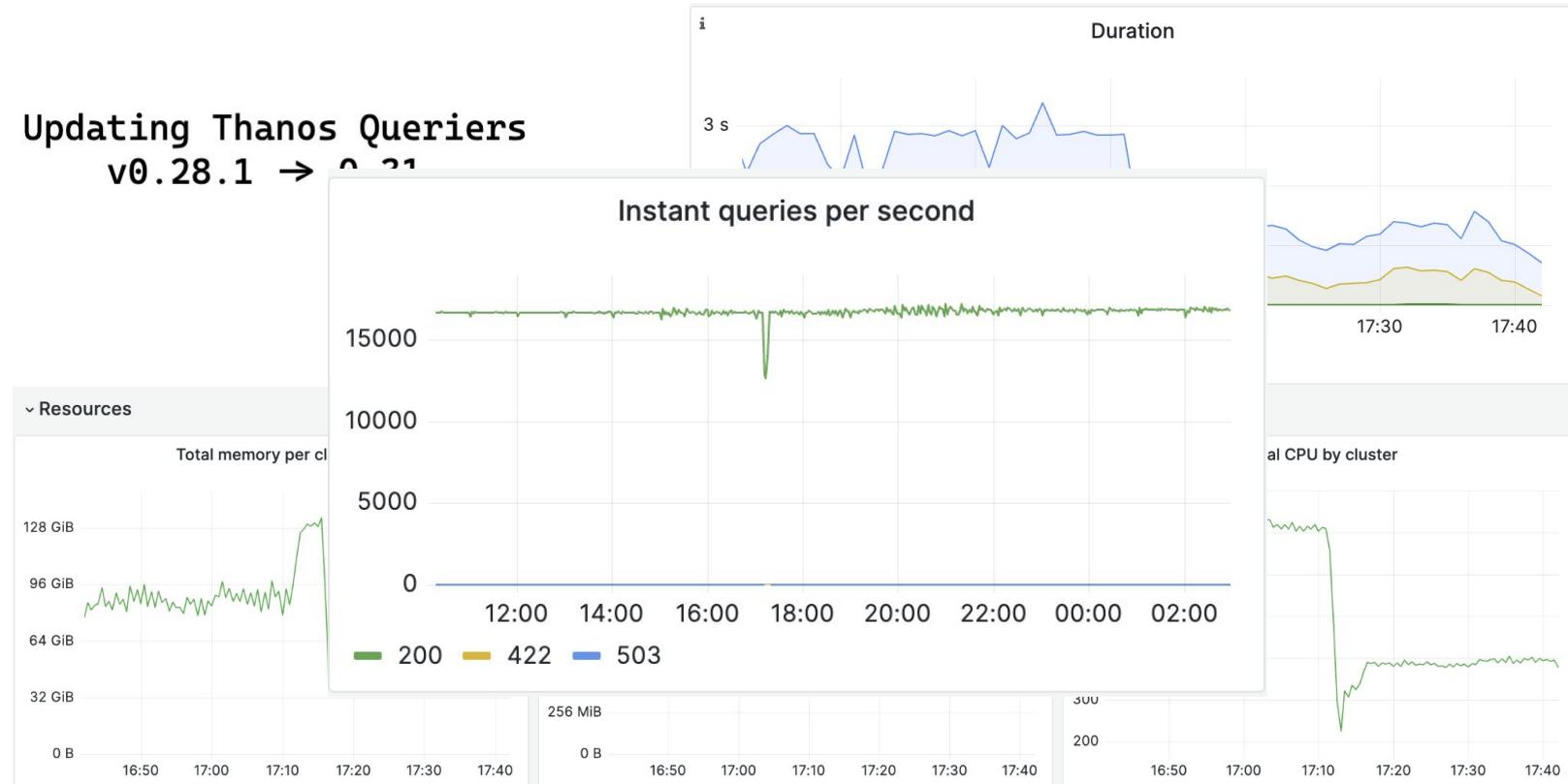
Does Thanos scale to X?

Head series



Constant improvements

Updating Thanos Queriers
v0.28.1 → v0.29.1



Thanos Community

How to get involved?

Thanos Community: Communication



Primarily happens on [#thanos](#) and [#thanos-dev](#) CNCF slack channels.

For code/bugs/issues/feature requests, GitHub issues/PRs on [Thanos](#) or other [thanos-io/thanos-community](#) repos.

Thanos Community: Meetings



Thanos Community Office Hours

Bi-weekly on Thursdays @ 2pm UTC

[Agenda Doc](#) + [CNCF Calendar](#) (not always correct!)

KubeCon [ContribFest](#) (or project meetings/booth)!

Thanos Community: Mentorships



<https://thanos.io/tip/contributing/mentorship.md/>

CNCF Linux Foundation Mentorships: Nearly every quarter

Google Summer of Code: Once a year

Past successful projects in: <https://github.com/cncf/mentoring!>

Thanos Community: Share your story!



Thanos Website has new blog space: <https://thanos.io/blog/>!

We'd love to hear from you!

Share how you are adopting or using Thanos, and get feedback from the community!



Thank You!

<https://thanos.io>