Sink or Swim: How not to drown in (colossal) streams of data?

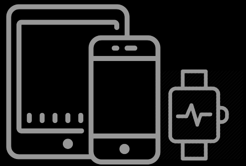
Nitin Agrawal

ThoughtSpot

“Colossal” streams of data

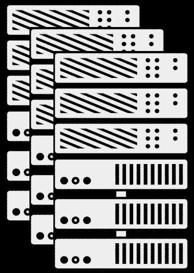
4 TB /car /day

x 100s thousands cars



10 MB /device /day

x millions devices



10 TB /data center /day

x 10s data centers

20 GB /home /day

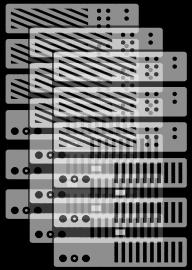
x 100s thousands homes

“Colossal” streams of data

4 TB /car /day
x 100s thousands cars

10 MB /device /day
x millions devices

Hundreds of **TB** to **PB** /day

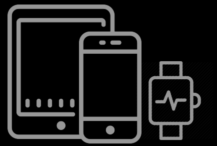


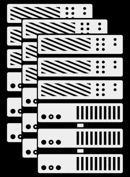
10 TB /data center /day
x 10s data centers

20 GB /home /day
x 100s thousands homes

Applications with “colossal” data

Need to support timely analytics





Analyses

- Forecast
- Recommend
- Detect outliers
- Telemetry
- Route planning

Applications with “colossal” data

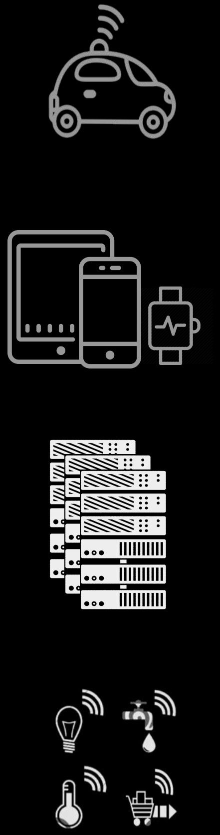
Need to support timely analytics

Analyses

- Forecast
- Recommend
- Detect outliers
- Telemetry
- Route planning

IoT Applications

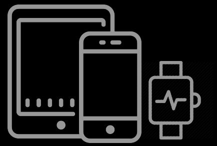
- Occupancy sensing
- Energy monitoring
- Safety and care
- Surveillance
- Industrial automation



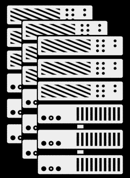
Applications with “colossal” data

Current solutions

In-memory analytics systems

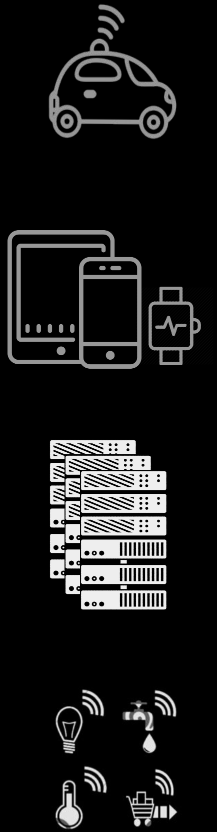


Conventional (storage) systems



Applications with “colossal” data

Current solutions



In-memory analytics systems

- Interactive latency, but $$$$
- Need secondary system for persistence

Conventional (storage) systems

DRAM "volatility"

Facebook and Amazon are causing a memory shortage

Demand for DRAM is soaring thanks to the explosive growth in hyperscale data centers, creating a shortage and causing prices to increase.

Dell EMC warns of server RAM price surge amid global shortage

iPhone 8 creating worldwide shortage of DRAM & NAND chips, says report

RAM prices could climb even higher thanks to datacentre demand

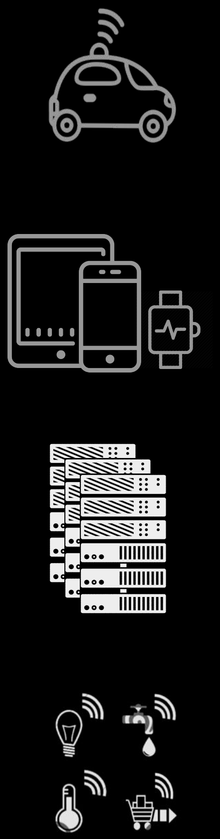
DRAM “volatility”

Covid-19 likely to send memory prices, markets down

DRAM spot prices rise 10-15% in September

Applications with “colossal” data

Current solutions



In-memory analytics systems

- Interactive latency, but $$$$
- Need secondary system for persistence

Conventional (storage) systems

- High latency
- Still quite resource intensive

I/O performance not keeping up

Improved dramatically over the years

But, still a bottleneck…

I/O performance not keeping up

Disk read performance (spec)

Random IOPS (4K)

- HDD: 61
- SSD: 400K

Sequential (MBps)

- HDD: 250
- SSD: 3400

Price

- HDD: $0.035/GB
- SSD: $0.5/GB

→

Query performance (spec)

	1 GB	1 TB
Random		
HDD	1 hr	48 days
SSD	0.6 secs	11 mins
Sequential		
HDD	4 secs	1 hr
SSD	0.3 secs	5 mins

Drowning in data

Continuous data generation on significant rise

- From sensors, smart devices, servers, vehicles, …
- Analyses require timely responses
- Overwhelms ingest and processing capability

Conventional storage systems can’t cope with data growth

- Designed for general-purpose querying not analyses
- Store all data for posterity; required capacity grows linearly
- Administered storage expensive relative to disks

Sink or Swim?

How not to drown?

Democratizing storage

- No one size fits all, store what the application needs.

Democratizing discovery

- Intuitive interfaces for end-users to engage with data.

How not to drown: democratizing storage!

Revisiting design assumptions around data

- Data streams unlike tax returns, family photos, documents
- Consumed by analytics not human readers
- Embracing approximate storage - not all data equally valuable for analyses

Applications designed with uncertainty and incompleteness

- Many care about answer “quality” and timeliness, not solely precision

Could store all data and lazily approximate at query time

- Slow: ingest and post-processing takes time
- Expensive: system needs to be provisioned for all ingested data

How not to drown: democratizing discovery!

Human-centric interfaces to data

- End users not always experts in query formulation.
- Embracing natural language querying and searching.

Custom data-centric applications without significant effort

- End users not necessarily have deep programming expertise.
- Empower writing new applications with low/no software development.

Embracing approximate storage

Proactively summarize data in persistent storage

- ▹ Fast: queries need to run on a fraction of data
  Summaries provide additional speedup
- ▹ Cheap: system provisioned only for approximated data
  Capacity grows sub-linearly or logarithmically with data
- ▹ Maximize utilization of administered storage and compute

Caveats and limitations of approximate storage

- ▹ Effectiveness depends on target analyses
- ▹ Interesting research questions!

Preview: potential gains with SummaryStore

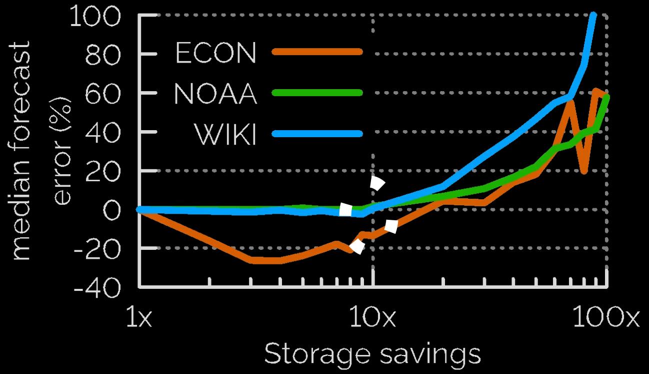
SummaryStore: approximate store for “colossal” time-series data

Key observation: in time-series analyses

- Newer data is typically more important than older
- Can get away with approximating older data more

In real applications (forecasting, outlier analysis, ...) and microbenchmarks:

scale	**1 PB** on single node (compacted **100x**)
latency	**< 1s** at 95th %ile
error	**< 10%** at 95th %ile



10x compaction
< 0.1% error

Forecasting

Challenges in building approximate storage

Ensuring answer quality

- Provide high quality answers under aggressive approx.
- Quantify answer quality and errors

Ensuring query generality

- Enable analyses to perform acceptably given approx. scheme
- Handle workloads at odds with approx. (e.g., outliers)

Reducing developer burden

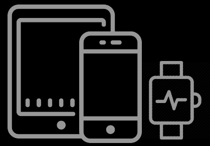
- App developers not statisticians; abstractions to incorporate imprecision
- Counter design assumptions across layers of storage stack

Applications with “colossal” data streams

Current solutions

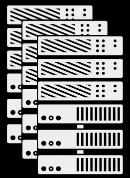
In-memory analytics systems

- Interactive latency, but $$$$
- Need secondary system for persistence



Conventional time-series stores

- High latency, still quite expensive



Approximate data stores?

- Promising reduction in cost & latency
- Current approx storage systems not viable for data streams

Goal: build a low-cost, low-latency store for stream analytics

Goal: build a low-cost, low-latency *approximate* store for stream analytics

Key insight

We make the following observation:

many stream analyses favor newer data over older
existing stores are oblivious, hence costly and slow

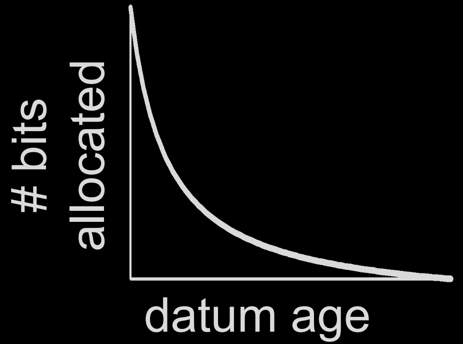
Examples:

Spotify, SoundCloud	Time-decayed weights in song recommender
Facebook EdgeRank	Time-decayed weights in newsfeed recommender
Twitter Observability	Archive data past an age threshold at lower resolution
Smart-home apps	Decaying weights in e.g. HVAC control, energy monitor

SummaryStore: approximate store for stream analytics

Our system, **SummaryStore***

- Approximates data leveraging observation that analyses favor newer data
- Allocates fewer bits to older data than new: each datum *decays* over time



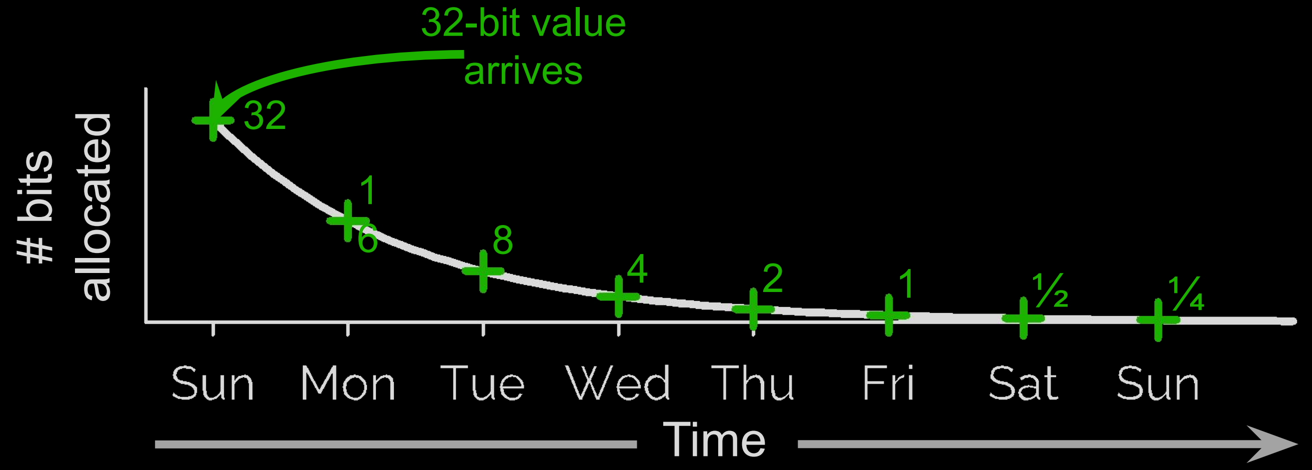
Low-Latency Analytics on Colossal Data Streams with SummaryStore, Nitin Agrawal, Ashish Vulimiri. SOSP ’17.

SummaryStore: approximate store for stream analytics

Our system, **SummaryStore**

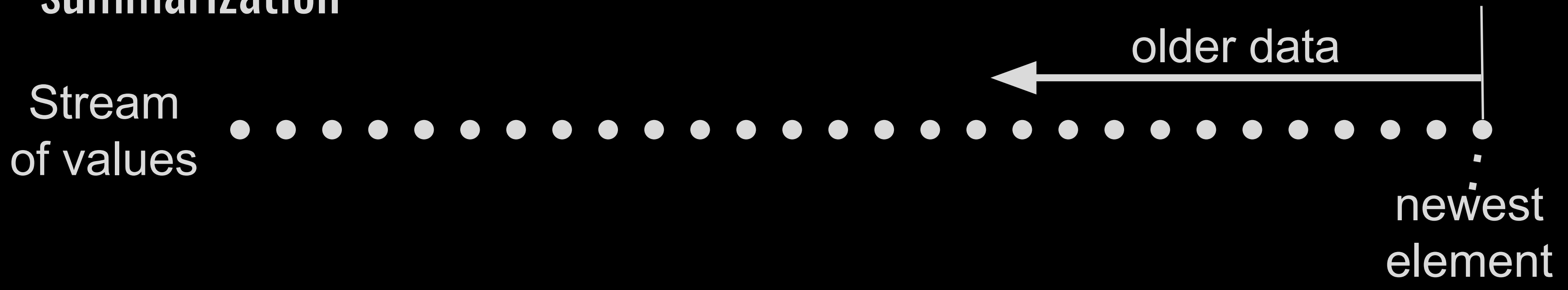
Allocates fewer bits to older data than new:
each datum *decays* over time

Example decay policy: halve number of bits each day



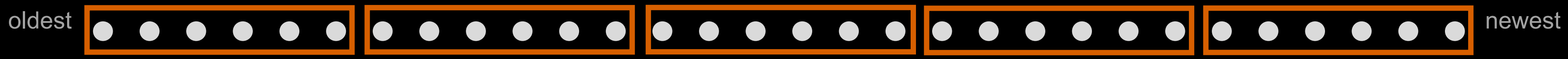
Time-decayed stream approximation

through windowed summarization



Time-decayed stream approximation

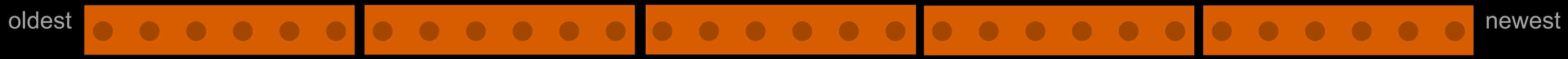
through windowed summarization



1. Group values in windows

Time-decayed stream approximation

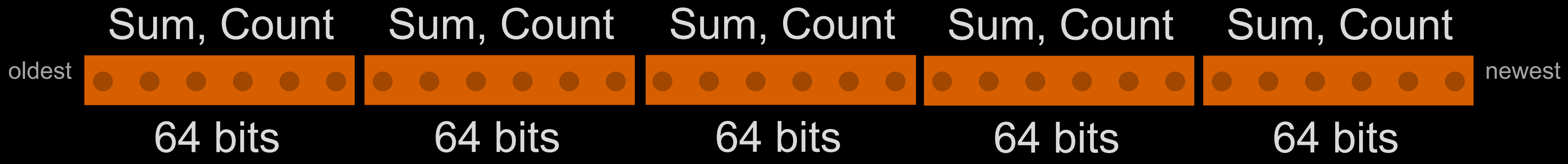
through windowed summarization



1. Group values in windows. Discard raw data

Time-decayed stream approximation

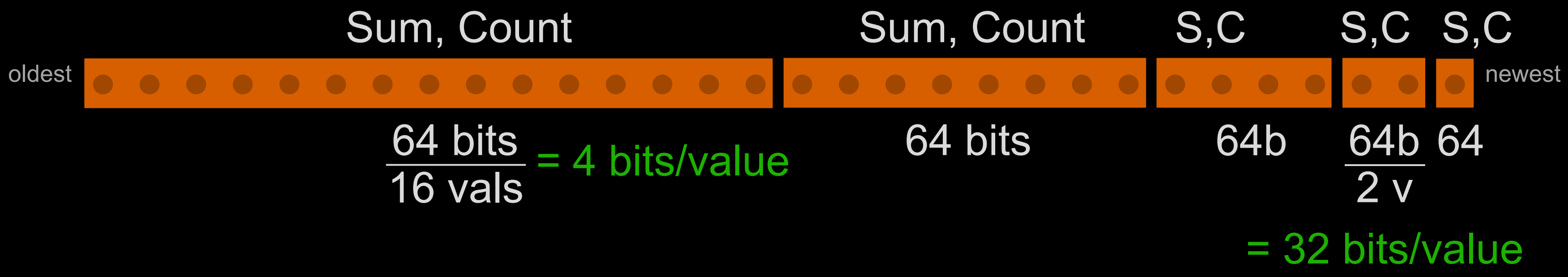
through windowed summarization



1. Group values in windows. Discard raw data, keep only window summaries
   - e.g. Sum, Count, Histogram, Bloom filter, ...
   - Each window is given same storage footprint

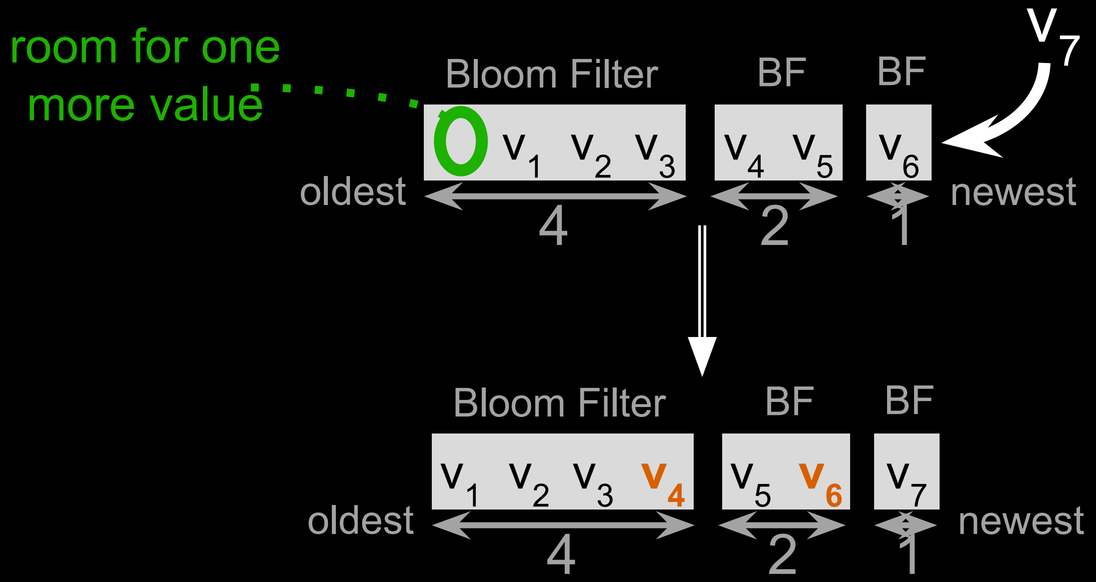
Time-decayed stream approximation

through windowed summarization



1. Group values in windows. Discard raw data, keep only window summaries
   - e.g. Sum, Count, Histogram, Bloom filter, ...
   - Each window is given same storage footprint
2. To achieve decay, use longer timespan windows over older data

Challenge: processing writes



Configuration:

Window lengths 1, 2, 4, 8, ...

Each window has Bloom filter

Don’t have raw values, only window summaries (Bloom filters)

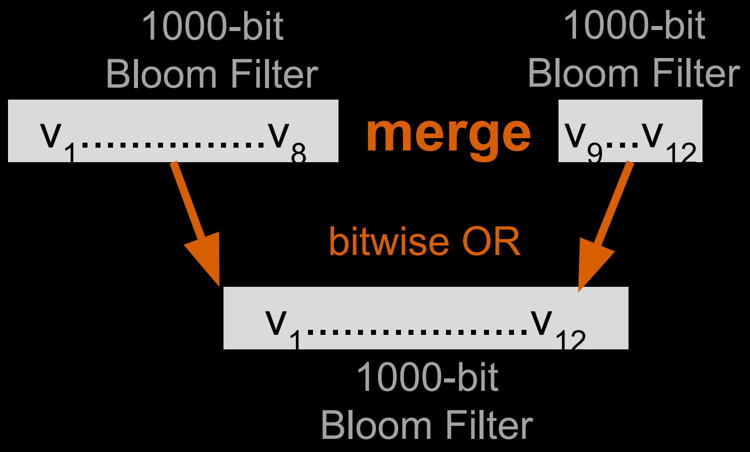
How do we “move” v_4, v_6 between windows?

Ingest algorithm

Not possible to actually move values

Instead, use a different technique, building on work by Cohen & Wang†

- Ingest new values into new windows
- Periodically compact data by **merging** consecutive windows
  - Merge all summary data structures



merge operation for

Bloom Filter	:	bitwise OR
Count	:	add
Histogram	:	combine & rebin

etc

† E. Cohen, J. Wang, "Maintaining time-decaying stream aggregates", J. Alg. 2006

Challenge: time-range queries

query a summary over
the time-range $[T_1, T_2]$

Examples

- What was average energy usage in Sep 2015?
- Fetch a random (time-decayed) sample over the last 1 year

Challenge: time-range queries

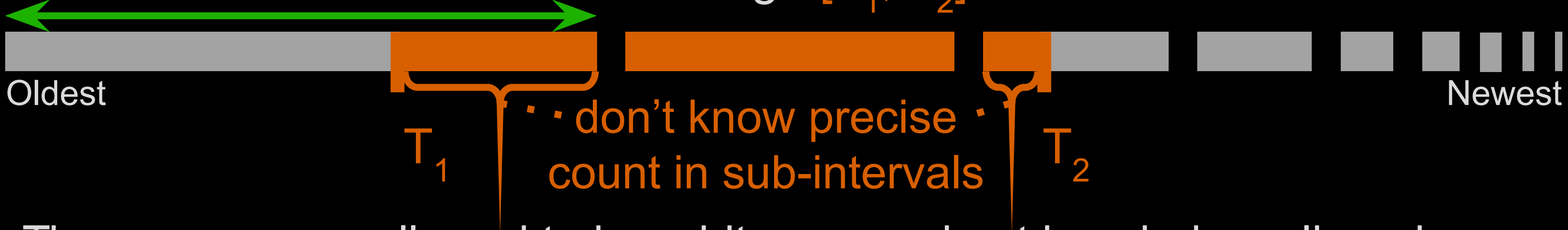
query a summary over
the time-range $[T_1, T_2]$

Time-ranges are allowed to be arbitrary, need not be window-aligned

Challenge: time-range queries

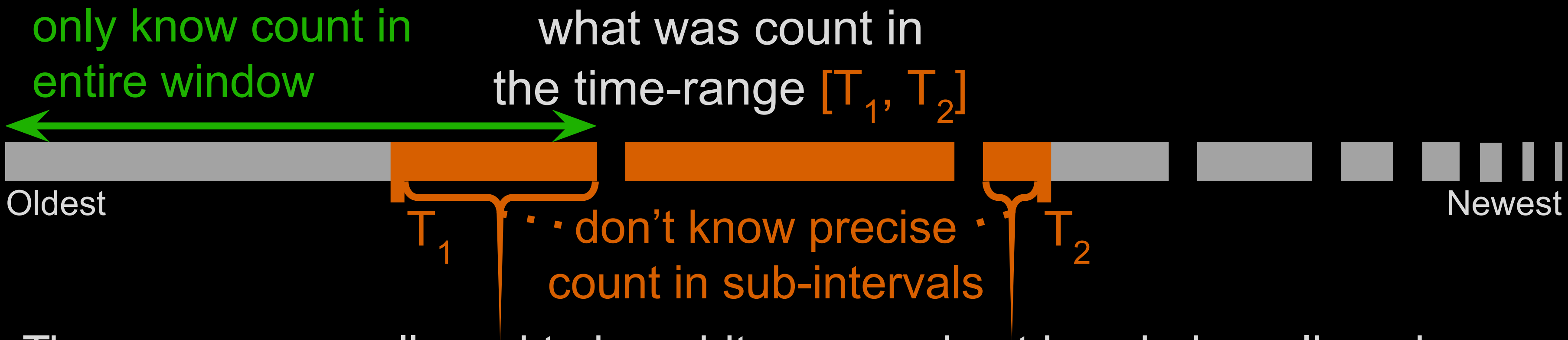
only know count in entire window

what was count in the time-range $[T_1, T_2]$



Time-ranges are allowed to be arbitrary, need not be window-aligned

Challenge: time-range queries



Time-ranges are allowed to be arbitrary, need not be window-aligned

Lack of window alignment introduces error

We use novel low-overhead statistical techniques to estimate answer & confidence interval

Query accuracy

Age = how far back in time query goes

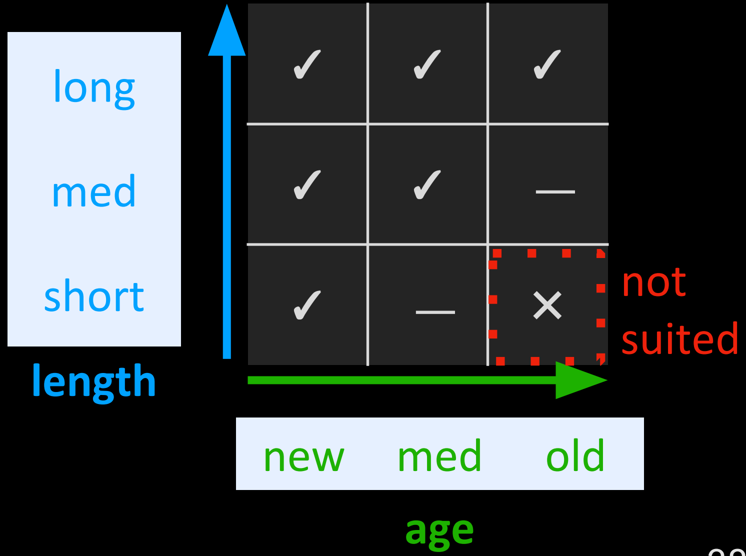
- Lower age ⇒ more recent data, so better accuracy

Length = time-span query covers

- Longer length ⇒ more windows spanned, so better

Not suited for large age + small length

- e.g. query over the time range
  [10 years ago, 10 years ago + 3 seconds]



Evaluation

On a single node: 224 GB RAM, 10 x 1 TB disk

Microbenchmarks: 1 PB on single node

Real applications

- Forecasting
- Outlier analysis
- Analyzing network traffic and data backup logs

Time-series forecasting w/ Prophet

Prophet: open-source forecasting library from Facebook

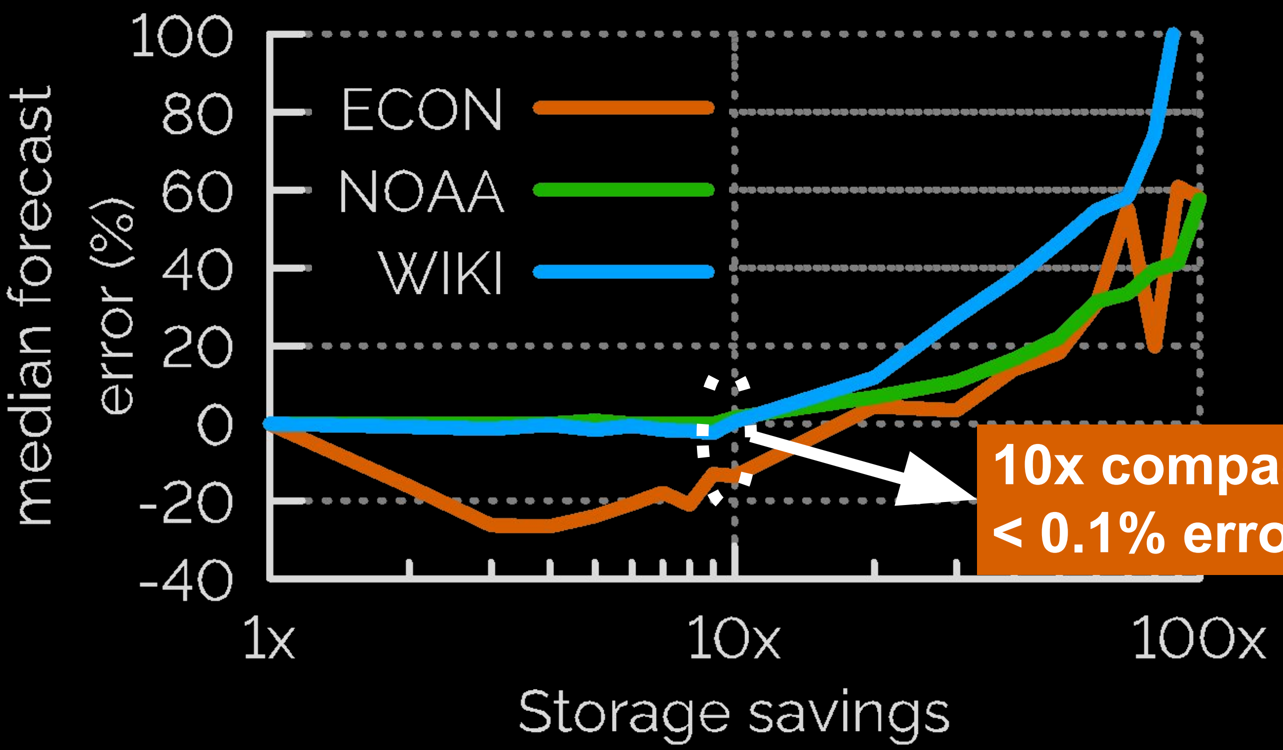
Tested three datasets

- WIKI: visit counts for Wikipedia pages
- NOAA: global surface temperature readings
- ECON: log of US economic indicators

On each time-series in each dataset, compared forecast accuracy of

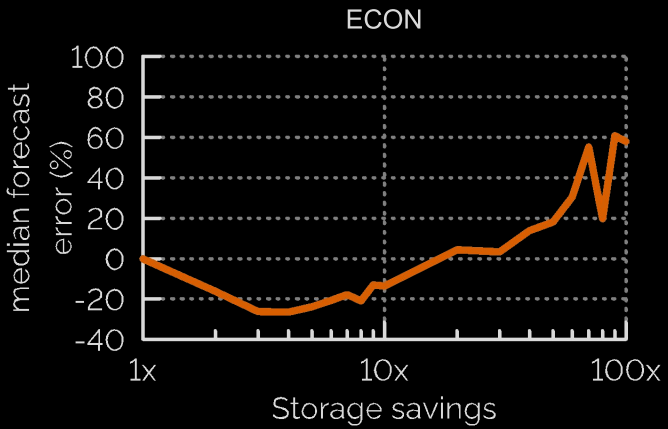
- Model trained on all data
- Model trained on time-decayed sample of data

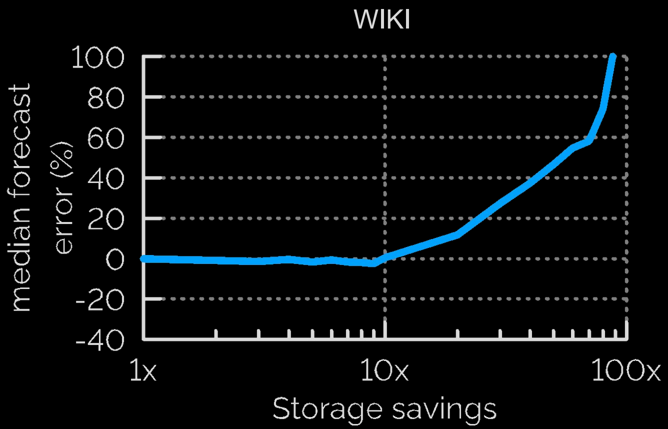
Time-series forecasting w/ Prophet

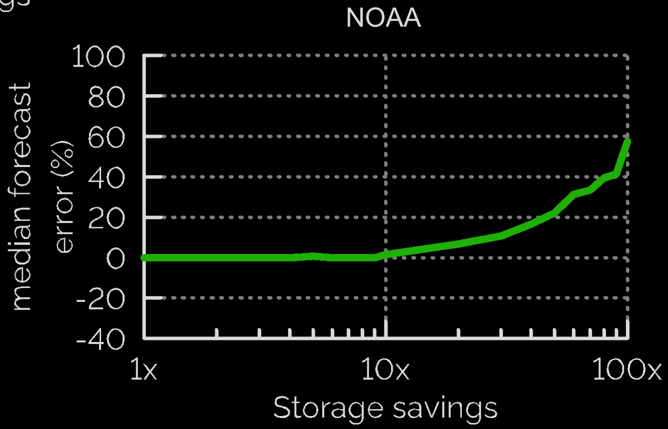


10x compaction
< 0.1% error

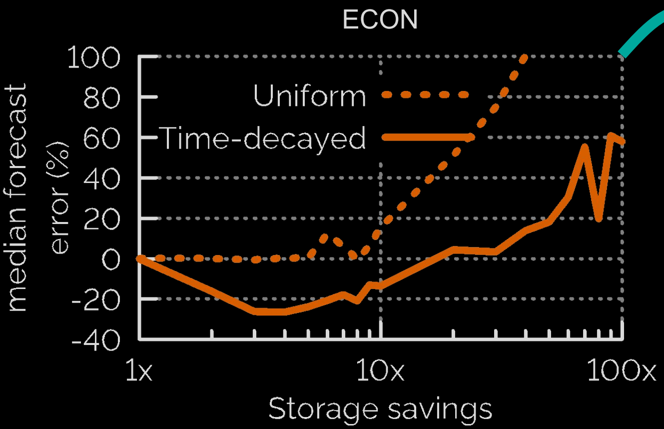
Time-series forecasting w/ Prophet



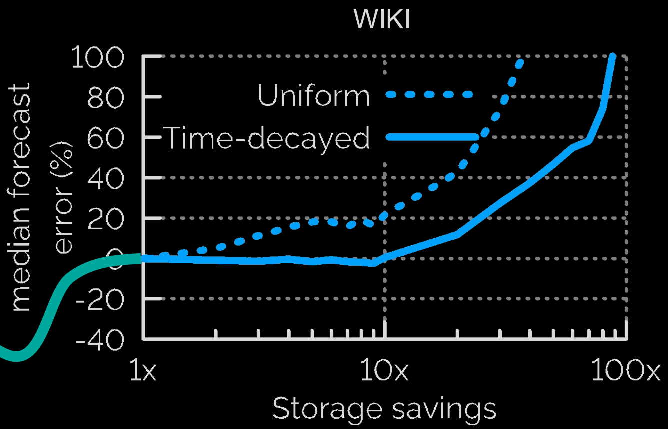


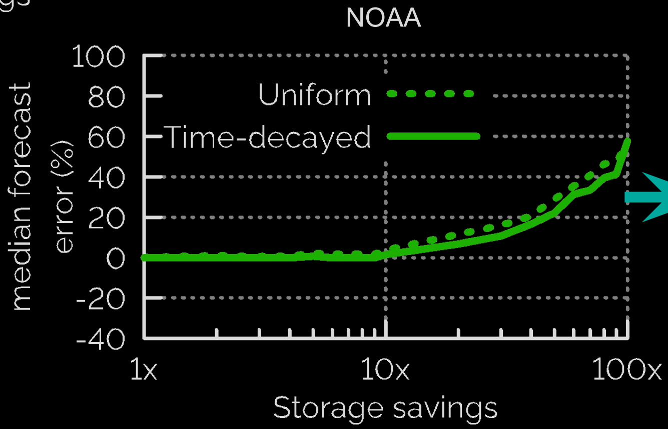


Time-series forecasting w/ Prophet



substantial improvement





difference not as stark because of predictable dataset

More details in paper

Landmarks

Ingest algorithm

System design

System configuration

Statistical techniques for sub-window queries

Landmarks

Mechanism for protecting specific values from decay

Values declared as landmarks are

- Always stored at full resolution
- Seamlessly combined with decayed data when answering queries

Example application: outlier analysis

Limitations

Choice of summaries needs to be defined a-priori at stream creation

Criteria for ”landmarks” also defined a-priori

- ▹ Scope of high-level analytics limited by the selection

Configuring rate of decay left to application

- ▹ Hard to estimate impact on individual query errors
- ▹ How aggressively can an application compact?

New summary operators can be added but require some effort

- ▹ Need to specify union function & model for error estimation

SummaryStore: approximate store for stream analytics

Contributions

- Abstraction: time-decayed summaries + landmarks
- Data ingest mechanism
- Low-overhead statistical techniques bounding query error

Works well in real applications and microbenchmarks:

- 10-100x compaction, warm-cache latency < 1s, low error
- 1 PB on a single node (summarized to 10 TB)

Project details and papers at https://bit.do/summarystore

Conclusions

Data streams everywhere, and growing

- ▹ Variety of analytics and learning apps require timely answers

Storage systems need orders of scaling to handle data growth

- ▹ Conventional approaches to scale up and scale out insufficient
- ▹ Conventional access paradigms increasingly insufficient

Broader research agenda around approximate computing

- ▹ Programming languages, architecture, user interaction, developer tools

New paradigms for data discovery and application development

- ▹ Human-centric interfaces to data siloed in storage systems

Thanks!