Integrated Real Time Multi-Dimensional Metrics & Near-Real-Time Analytics

Jordan Edwards

1 Introduction

This document describes a real-time metrics framework used today by the Application Host in Microsoft's Applications and Services group – it involves leveraging our rapid-query metrics data store (MetricSystem) alongside other complementary metrics technologies (such as ElasticSearch and Hadoop).

The following are high-level goals of a real-time metrics & insights infrastructure:

- Query time-series data as it is being ingested for both immediate and historical insights.
- Aggregate, drill-down, and slice-n-dice N-dimensional data with consistent, fast response times.
- Scalable and available; consistent and predictable performance in multi-tenant environments.

2 Real Time Metrics

2.1 Important Characteristics of Real Time Metrics

In our multiple years of working on performance oriented distributed services, we found that the vast majority of engineers leverage fast metric solutions to give them insights into the behavior of their applications.

We set out to find a complementary set of offerings that would enable real-time application insights w/ minimal overhead & turnaround time. Here are some targets we designed for (based on the needs of our specific system):

- Real-time
    - Query time: ≤1s for hour-over-hour comparison data
    - EventOccurs-to-Queryable: ≤5min (w/ a target of ≤ 1)
- Highly dimensional
    - Permutations per-metric: 1M (supported)
    - Native support for ad-hoc queries (rapid rollout across non-filtered dimensions)
- Highly available
    - 99% availability - environment-level data
    - Tune-able per-machine data availability (optional persistence layer)

2.2 Existing solutions

There are multiple existing metrics solutions which exist external to the company. The most popular of these is Graphite, a legacy performance counter system leveraged across many external companies – however, it does not support the level of permutations or scale of data we were built to ingest.

2.2.1 InfluxDB

InfluxDB (which is backed by LevelDB) provides a similar set of features to consumers, but is not optimized for our core scenarios. The query language also has special features that will likely make it possible to work with large numbers of series fluidly. By that, I mean a query that wants to operate over a million series doesn't have to mention them all by name. Pattern matching on series names is an important part of this. InfluxDB is inherently a distributed database – it does not function well as a standalone metrics agent.

2.2.2 Druid

Druid is an open source data store designed for real-time exploratory analytics on large data sets. The system combines a column-oriented storage layout, a distributed, shared-nothing architecture, and an advanced indexing structure to allow for the arbitrary exploration of billion-row tables with sub-second latencies.

It is perhaps the closest real-world analog to the system we built, with a few key differences:

1. Each MetricSystem agent acts as a complete metrics store solution (Druid relies on a set of real-time and historical nodes with state management in between for query resolution)
2. MetricSystem only supports numeric value types

2.3 MetricSystem - A highly scalable, on-machine metrics agent

MetricSystem is a distributed, column-oriented, real-time analytical data store. It has been purpose-built to power insights for high performance applications and is optimized for low query latencies (both in terms of time to query, as well as time to ingest) and minimal network transit (it functions as a standalone on-machine metrics agent.) It takes advantage of numerous performance optimizations available in the CLR and is fully managed code.

MetricSystem can quickly ingest massive quantities of multi-dimensional time-series data and make that data immediately available to queries. It is not a database – it is an append-only datastore optimized for rapid lookup of multi-dimensional data.

The need for MetricSystem was facilitated by the fact that existing RDBMS / NoSQL metrics solutions were unable to provide a low latency data ingestion and query platform for the highly dimensional metrics produced by services such as Bing's Application Host.

We built for rapid querying, massive dimensionality, and high availability (such that we are able to perform A/B testing on the performance of application code as the application is rolling out).

2.3.1 Some numbers

MetricSystem collects 100MB/min/machine of telemetry data, which spans across 100 separate multi-dimensional metrics. This processing typically happens in seconds. Once the data is available to query, results can be returned on the order of milliseconds. This service is running across 10,000 machines globally and is tune to leverage <1 GB of memory and a single core per-machine (this is obviously customizable based on scenario). This equates to a globally queryable amount of 200TB/hr.

3 MetricSystem Components

3.1 Real-time event ingestion endpoints

We expose a real-time event listener, both over ETW and via a RESTful /write API. This API is capable of at least 20,000 writes per second (this figure is for REST – ETW API results TBD, but are expected to be notably better).

3.2 Storage Engine: Data Tables (.msdata)

Data tables in MetricSystem (which we represent as MSDATA files) are collections of timestamped events and partitioned into a table per metric, where each table contains ~1M independent metrics.

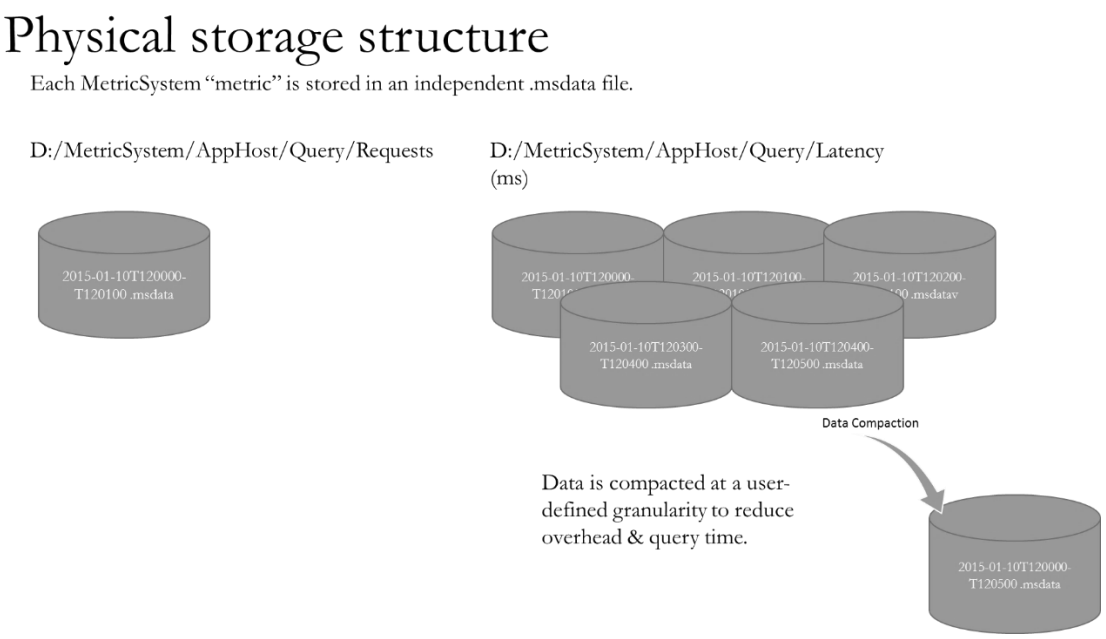
Each data table is segmented based on timestamp, with a metric name and time bucket serving as a composite key for retrieving the table.

3.2.1 Types of storage structures

1. As events flow into the system, they are written to an in-memory, write-optimized data structure (these are considered "real time tables")
2. Once memory constraints have been hit, a real-time table is persisted to a compacted binary collection, optimized for the ability to rapidly load the data back into memory for querying (without causing large amounts of large object heap allocation in the CLR).

MetricSystem's data ingestion focuses on rapid ingestion and aggregation of individual time series events (it is append-only and optimized for performance counter scenarios).

3.2.2 Physical storage structure format



Physical data is compacted at a user-defined granularity to increase available time window storage and reduce query time (less seeks from the SSD are required).

3.3 Query Service

MetricSystem has its own query language and accepts queries as GET or POST requests.
Each MetricSystem query agent can act in one of two modes:

1. Local agent mode, accessing localized query data (from in-memory or on-platter tables)
2. Aggregator mode, fanning out requests to intermediates, and aggregating tiers of data that comes back.

The body of the POST request sent to an agent is a JSON object containing key-value pairs specifying various query parameters.

A typical query will contain the data source name, the granularity of the result data, the time range of interest, the type of request, and the metrics to aggregate over. We also support a single dimensional split today.

3.3.1 Query API Examples

- **/write**
    - List<CounterWriteOperation>
    - Example: [{ counterName: "foo", dimensions: {}, value: x, count: y }]
    - For both hit count and histograms
- **/info** - Returns a list of available counters for the agent you are querying, with their respective dimensions
    - <param name="counterPattern"> Pattern to match counter names against (glob style).</param>

    - <param name="queryParameters"> Optional parameters for the query.</param>
    - <returns> A CounterInfoResponse object with all retrieved information.</returns>

- **{counterName}/query -** Returns data from N buckets, based on certain filters (start=foo, end=bar, dimensionA=baz)
    - <param name="counterName">Name of the counter to query.</param>
    - <param name="server">Server(s) to query.</param>
    - <param name="queryParameters"> Optional query parameters for the counter.</param>

The result will also be a JSON object containing the aggregated metrics over the time period (transferred over the wire as JSON or compacted binary).

3.3.2 Full list of supported operations

- List metrics
- List timestamps per metric
- List dimensions per metric
- List values per dimension
- Query data for time window (SELECT LATENCY FROM MSDATA)
    - WHERE
    - ORDER BY (in progress)
    - LIMIT N (in progress)

3.4 Real-Time Aggregation

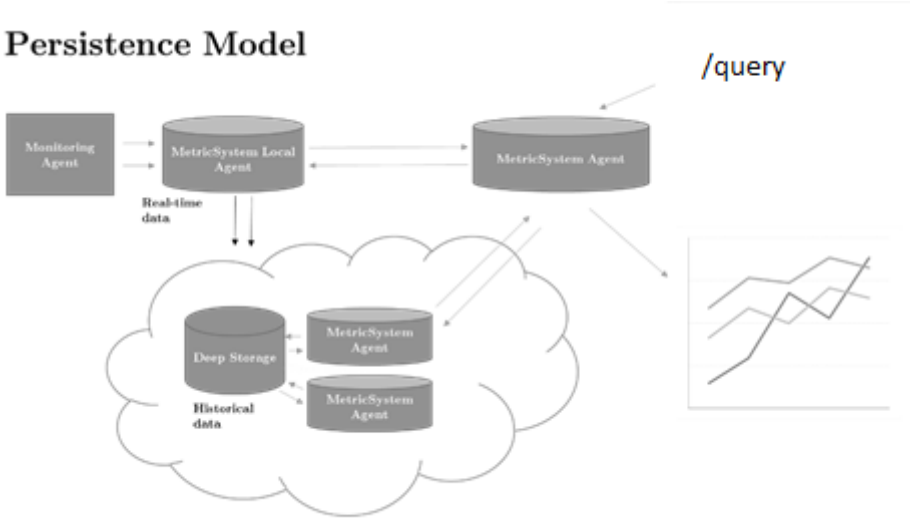
MetricSystem performs tiered, real-time aggregation similar to Facebook's SCUBA (through our aggregation client, which we call the Kraken). It federates requests out to intermediate tiers of machines to perform work (unwrapping average operations for intermediate steps to avoid loss of data fidelity). Aggregated data is then provided back to the user and cached on the current agent.

3.4.1 Aggregation Process

1. Query goes from client to root aggregator node
2. Root aggregator finds N intermediate aggregators via **registry service** (or by specifying hosts in the envelope).
3. Any average functions are replaced with sum / count.
4. Query is sent to N intermediate aggregators (any client can be an aggregator node).
5. Intermediate aggregators fan out the query again to leaf aggregators (one per pod?)
6. Leaf aggregators collect data from leaves.
7. Leaf aggregators push data to intermediate aggregators.
8. Intermediate aggregators push to root, which returns response to the client.

3.5 Persistence Model

Each MetricSystem agent has configurable resource utilization. Deep storage can be represented on-machine (SSD) or in Azure. MetricSystem agents run localized against data to allow for real-time aggregation across both fresh and historical data. Figure 1 outlines our persistence model.



4 Complementary technologies to MetricSystem

4.1 Messaging Queues

In order for messages to be ingested into a near-real-time framework, they need a temporary place of residence. These technologies are typically referred to as messaging queues (of which Apache Kafka is perhaps the most well-known example).

Messaging queue technologies are fairly straightforward and well documented externally, so we will not delve into them in this paper. Some brief synopses are below.

4.1.1 Apache Kafka

Kafka is a distributed, partitioned, replicated commit log service. It provides the functionality of a messaging system, but with a unique design. Several teams are hosting and leveraging Kakfa internally (the Veritas pub/sub system is leveraging it inside of the Shared Data Platform team).

Kafka maintains feeds of messages in categories called topics. Processes that publish messages to a Kafka topic producers. Processes that subscribe to topics and process the feed of published messages consumers. Kafka is run as a cluster comprised of one or more servers each of which is called a broker.

4.1.2 Azure EventHubs

Azure offers a PaaS messaging queue solution called EventHubs and many teams internally have built competitive solutions to address their needs.

4.2 Stream Processing

Stream processors (such as **Storm, Baja** and **Orleans**) are capable of providing real-time MapReduce functionality on data as it flows through a metrics pipeline. This can be useful in filtering, dimensionality, anomaly detection, as well as annotation of additional metadata onto incoming messages.

4.2.1 Storm

Storm is a stream processing framework originally developed by Twitter. Storm makes it easy to reliably process unbounded streams of data, doing for realtime processing what Hadoop did for batch processing. Microsoft offers an internal .NET wrapper on top of Storm called SCP.NET (also referred to as Tachyon).

4.2.2 Orleans

Orleans is a distributed computing framework developed by Microsoft Research. It abstracts away the need to deal with distributing your service across an application fabric using task-oriented Actors (which it describes as grains) working inside of an application boundary (or silo). Each grain is instantiated as a stateful or stateless worker object, allowing this system to be customized to provide stream computing for teams such as Halo and Skype (who place a high value upon rapidly actionable insights).

4.3 Full-Text Search

By design, MetricSystem only stores processed quantitative data (that is, either rate counters represented as integers, or histograms represented as integer dictionaries).

As a complementary offering, we found that the optimal way to enable NRT analysis of text-based search data was to leverage a hosted ElasticSearch offering.

4.3.1 ElasticSearch

ElasticSearch is a highly performant, Lucene-based analytical search engine. By integrating ES offerings alongside MetricSystem, we offer near real-time solutions for both quantitative and qualitative data.

ElasticSearch offers a three tiered stack – LogStash (for log processing and pushing), ElasticSearch (for storage and querying of these logs), and Kibana (for visualization and the creation of diagnostic dashboards).

4.4 Cold Storage / Distributed Analytics Platforms

Teams across Microsoft typically leverage one of two distributed MapReduce frameworks - **Cosmos** or **Hadoop**. We will not delve too deeply into these platforms, but note that they are both widely used inside of Microsoft and in the wider technology industry. A common pattern we find ourselves performing is "cooking"

large amounts of raw log data to create historical reports (while we previously did this leveraging SQL Server, we are moving more and more into a document-store oriented visualization framework).

5 Making Data Useful

One of our current goals is providing a unified data access and visualization framework to make engineers able to effectively use the information they are producing across the many services they work on.

The below sections outline what we view as the key facets of "metrics user experience".

5.1 Discovering what data is available

Exploration and discovery is a key facet of livesite investigations. Users should be able to quickly construct builders from relevant data streams and indices for use in visualization.

Here are some example data discovery scenarios:

- Users need the ability to drill into different datastreams, spot anomalies.
- Connect relevant Search log streams together from different levels of the service stack.

5.1.1 Creating visualizations

Visualizations (and the data underneath them) need to effectively describe the insights which are hidden inside of the data. Main goals from a querying and visualization experience would be to provide a visualization format which can describe data in a seamless and unified fashion (across all discovered data streams).

Here are some data visualization scenarios:

- Query and filter functionality – Queries and filters should be constructed, with the ability to dynamically pass these queries and filters across additional visualizations (via pinning).
- Pivot visualization and data type on the fly – Each chart should be dynamically configurable to enable users to pivot which underlying data streams and being used, as well as which visualization is being used to best represent the data.
- Create metrics mashups – Ability to mashup data from different source into insightful visualizations.
- Build comparison scorecards which integrate statistical analysis frameworks (for things such as anomaly detection).

5.1.2 Building dashboards

A main goal in building performance dashboards is enabling metrics and KPI in interactive and performance charts and grids. The following scenarios should be natively supported in a dashboarding experience:

- Dynamic drilldown (visualizations should have inter-operability with cascading user experiences)
- Drill-through into historical data
- Put data in a table
- Easily spot topN outliers

- Find specific issues occurring to a service in a given timeframe
- Decorate existing data feeds to enable users to mark specific hot-points for follow up.
- Rapidly create alerts on top of datastreams

5.1.3 Building intelligence

Once we have figured out the data that is important / relevant, extend these queries into an existing anomaly detection framework. We are leveraging a number of node.JS packages to become more intelligent and more autonomous in the ways we interact with metrics data on a daily basis.

6 Additional Resources

- http://slides.com/torkelo/devsum-metrics-and-logging - walkthrough of DevOps scenario w/ ElasticSearch and Graphite
- http://techblog.netflix.com/2013/12/announcing-suro-backbone-of-netflixs.html Netflix's Suro pipeline (similar to our pipeline)
- https://facebook.com//download/600610293294881/scuba-vldb2013.pdf Scuba (Facebook's metric system)
- http://research.google.com/pubs/archive/36632.pdf Dremel (Google's metrics system)
- http://vldb.org/pvldb/vol5/p1436_alexanderhall_vldb2012.pdf PowerDrill (Google's stream pre-processing platform)
- http://orleans.codeplex.com (Orleans distributed computing fabric)
- https://storm.apache.org (Homepage for Apache Foundation Storm)