GIGASPACES
SINGLE PLATFORM. COMPLETE SCALABILITY

Real-Time Analytics for Big Data

An Alternative Approach

July 2011

Introduction

Real-time analytics is a hot topic, especially since Facebook published how they designed and implemented their real-time analytics system. [1] Facebook posited several design assumptions that centered on the reliability of in-memory systems and database neutrality that affected what they did, including: Transactional memory is unreliable, and HBase as the only targeted data store.

Can we challenge these assumptions? Reliable transactional memory exists in the field, it is a requirement for any in-memory data grid; and there are certainly more databases than only HBase. Given database and platform neutrality, and reliable transactional memory – what kind of real-time analytics system could be created?

Executive Summary

Real-time analytics are becoming part of mainstream system design, with high-profile companies such as Facebook sharing their design and implementation processes, proving that real-time is already a reality.

However, most of these designs rest on assumptions that inherently limit the resulting systems, among t hem the idea that memory is unreliable, and that there is only one choice of database.

This paper examines the proposition that these assumptions should be challenged, and that by changing them, inherent limitations of real-time analytics systems can be eliminated.

Real-Time Analytics – What's Been Done

To understand what a new design can look like, it might be useful to consider a previous design. Here is a short summary of Facebook's real-time analytics system, to which we will compare our alternatives.

The Facebook solution is based on a system of key/value pairs, where the key can be a URL and the value is a counter. Thus, there's a requirement for atomic, transactional updates to a very simple piece of data. The difficulties come from scale, not from the focus of the system.

The process flow is relatively simple:

- A user creates an event by performing some action on the website. This generates an AJAX request, sent to a service.
- *Scribe* is used to write the events into logs, stored on HDFS.
- *PTail* is used to consolidate the HDFS logs.
- *Puma* takes the consolidated logs from PTail and stores them in HBase in groupings that represent approximately 1.5 seconds' worth of events.
- *HBase* serves as the long-term repository for analytics data.

There are some questions around how PTail and Puma serve as scaling agents, and some of the notes around their use are still limited in scale – for example, one of the concerns is that an in-memory hash table will fill up, which sounds like fairly serious limitation to have to keep in mind.

[1] See also Nati Shalom's (GigaSpaces CTO) blog post on Facebook's real-time analytics for big data

The Potential for Improvement: How Can It Be Done Better?

There are many areas with potential for improvement, if the assumptions are changed. As a contrast to Facebook's working system:

- The design can be simplified. If memory can be seen as transactional – and it can – it can be used without transformation as it proceeds along the analytics workflow. This makes design and implementation much simpler to implement and test, and performance improves as well.
- The design can be strengthened. Systems using a polling semantic are brittle, relying on systems that pull data to generate real-time analytics data. The fragility of the system can be reduced, even while making it faster.
- The implementation can be strengthened. Batching subsystems create limits that need not exist. For example, a concern in Facebook's implementation is the use of an in-memory hash table that stores intermediate data; the in-memory aspect is not a concern until you realize that the batch sizes are chosen partially to make sure that this hash table does not overflow the available space.
- Deployments can switch databases based on their requirements. There is nothing wrong with HBase, but it has specific characteristics that are not necessarily a good fit for all enterprises. It is possible to design a system that can be deployed on various and flexible platforms, and the underlying long-term data store can be migrated to a different database if necessary.
- The analytics system can be consolidated so that management is easier and unified. While there are system management standards like SNMP that enable management events to be presented in the same way regardless of their source, having so many different pieces means that managing the system requires an encompassing understanding, which makes maintenance and scaling more difficult.

What we want to do, then, is create a general model for an application that can accomplish the same goals as Facebook's real-time analytics system, while leveraging the capabilities that in-memory data grids offer, for possible improvement in areas such as scalability, manageability, latency, platform neutrality, and simplicity – all while increasing ease of data access.

It sounds like quite a tall order, but it *is* doable.

The key is to remember that, at heart, real-time analytics systems are events systems. Facebook's entire architecture is designed to funnel events through various channels, so that Facebook's operations can safely and sequentially manage event updates.

Thus, as an analogy -- they receive a massive set of events that "look like" marbles, which they line up in single file; they then sort the marbles by color, you might say, and for each color they create a bundle of sticks; the sticks are lit on fire, and when the heat goes up past a certain temperature, steam is generated, which turns a turbine.

It's a real-life Rube Goldberg machine, which is admirable in that it works, but much of it is still unnecessary if the assumptions about memory ("unreliable") and database ("HBase is the only target that counts") are changed. Going back to the previous analogy, there's no need to change a marble into anything. The marble is enough.

A Plan for Implementation

Our design for implementation is built around putting data and messaging together. A data grid is a perfect mechanism for this, as long as it provides some basic features: transactional operations, push and pull semantics, and data partitioning.

And a data grid *does* provide those basic features, or else it's not really much of a data grid; it'd be more of a cache otherwise.

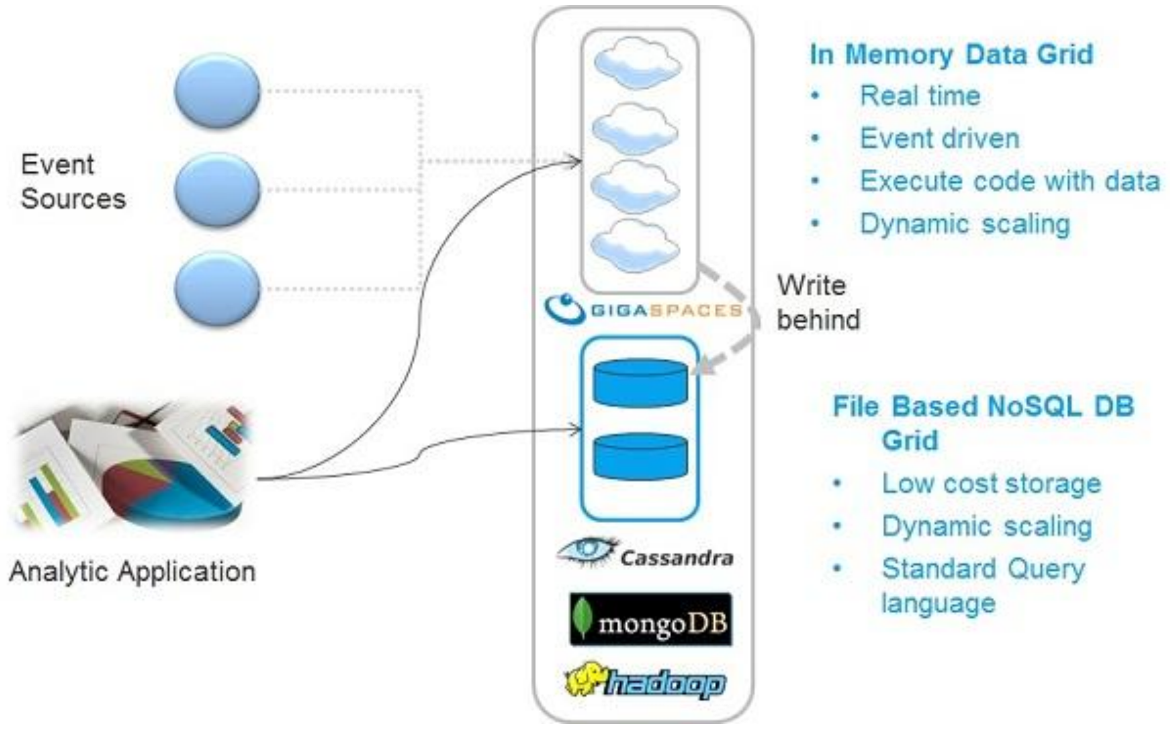
With a data grid, then, the events come in as individual messages. In our system, when the user chooses an operation on the web site, an asynchronous operation would write the event, just as Facebook does today. However, instead of filtering and batching the events into various forms, the events are dispatched to waiting processes that perform multiple transactional updates in parallel.

It is possible that these updates might be slower than the generated events, if each event is processed sequentially. That said, it isn't really much a problem: if data partitioning is used, then event handlers can receive partitioned events, which localizes updates and speeds them up dramatically.

In fact, you can still use batching to process events as a group; because the events would be partitioned coming in, the batch process would still be updating local data very quickly, which would be faster than individual event processing, while still retaining simplicity.

With this design there is no overflow condition, because a system designed to scale in and out – like most data grids – repartitions to maintain even usage. If a data grid does not provide this feature intrinsically, some management would be necessary. However, it is not difficult to find a data grid with this feature.

One other advantage of data grids is in write-through support. With write-through, updates to the data grid are written asynchronously to a back-end data store – which could be HBase (as used by Facebook), Cassandra, a relational database such as MySQL, or any other data medium you might choose for long-term storage.



The memory system and the database – the external data store – work together. The in-memory solution is ideal for the real-time aspects, the events that affect *now*. The external data storage solution is designed to handle long-term data, for which speed is not as much of an issue.

The Strengths of the Alternative Approach

The key concept here is that event handling is the lever that can move the real-time analytics mountain. By providing a simple, scalable publisher/subscriber model, design is simplified; by using a platform that supports data partitioning, transactional updates, and write through capabilities, scalability is achieved.

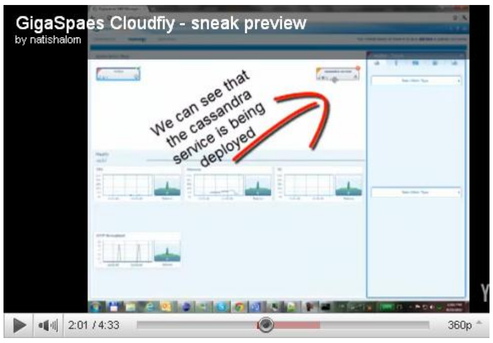
The data grid's flexible query API means that events can literally react when data is available.

For example, a call center might want to immediately identify signals that indicate that a caller should be handled differently; or, imagine an e-commerce site that is able to determine immediately if a user loses interest, and can respond appropriately, before the customer moves on.

With external processes and a long funnel for data, immediate-response capabilities are very difficult to implement, not only because of latency but because the data transformations tend to homogenize the data, instead of allowing rich expressions and flexible event types.

The data grid also has much richer support in terms of client applications. Instead of applications going through an API that focuses on a specific phase of the data’s life (for example, an API focused on HBase), you can focus on a generic API that captures events at any point in their life cycle, and from anywhere. An external monitoring process, then, can have the same immediate, partition-aware access to data that the integrated message-handling system does; adding features and analysis is just a matter of connecting a client to the data grid.

You can see a demo of these ideas put into action here. It shows how a market analysis application is deployed onto GigaSpaces XAP through our new Cloud deployment tool, using an event-driven system to display real-time data, with a write-through to Cassandra on the back end. The design is very simple, and demonstrates the principles discussed here – and it can scale up and down depending on demand.


GigaSpaes Cloudfiy - sneak preview
by natishalom
We can see that the cassandra service is being deployed
2:01 / 4:33
360p

Conclusion

As real-time analytics gets into mainstream application design, it becomes important to draw a blueprint on how to build this sort of system without reinventing the wheel.

Todd Hoff (HighScalability) and Alex Himel (Facebook) provided detailed descriptions of their solutions and even more importantly, they shared the rationales for doing what they did.

One main difference between their assumptions and ours, leading to the different implementation strategies, are those regarding reliable memory for event processing, and the use of passive data storage.

Another difference is that at GigaSpaces we had to think of the solution as an easily clone-able solution, and therefore a lot of attention was put on the simplicity of the runtime, packaging, and management of the solution.

Yet another difference is that we did not choose one specific database, as there is no “one size fits all" solution. Some customers, for example, still prefer SQL, and the fact that we can buffer the write to the database gives them more headroom while still allowing them to scale on writes.

About GigaSpaces

GigaSpaces Technologies provides a new generation of application virtualization platforms. Our flagship product, eXtreme Application Platform (XAP), delivers end-to-end scalability across the entire stack, from the data all the way to the application. XAP is the only product that provides a complete in-memory solution on a single platform, enabling high-speed processing of extreme transactional loads. XAP was designed from the ground up to support any cloud environment – private, public, or hybrid – and offers a pain-free, evolutionary path from today’s data center to the technologies of tomorrow.

Hundreds of organizations worldwide are leveraging XAP to enhance IT efficiency and performance. Among our customers are Fortune Global 500 companies, including top financial services enterprises, telecom carriers, online gaming providers, and e-commerce companies.

U.S. Headquarters
GigaSpaces Technologies Inc.
317 Madison Ave, Suite 823
New York, NY 10017
Tel: 646-421-2830
Fax: 646-421-2859

U.S. West Coast Office
GigaSpaces Technologies Inc.
101 Metro Drive, Suite 350
San Jose, CA 95110
Tel: 408-878-6982
Fax: 408-878-6149

International Office
GigaSpaces Technologies Ltd.
4 Maskit St., P.O. Box 4063
Herzliya 46140, Israel
Tel: +972-9-952-6751
Fax: +972-9-956-4410

Europe Office
GigaSpaces Technologies Ltd.
2 Sheraton St.
London, W1F 8BH, United Kingdom
Tel: +44-207-117-0213
Fax: +44-870-383-5135