

Jonathan Pinnock and Associates

Software Development, Consultancy and Some Pretty Cool Products

White Paper: Monitoring Market Data Latency and Quality

Revision 1.1
26 September 2006

www.jpassoc.co.uk

This white paper describes how the various tools in the JPA PlatformOne™ product suite can be used to monitor the effectiveness of contributed data. There are two aspects to making contributed data more effective: minimizing its latency, and maximizing its quality.

Tracking Latency

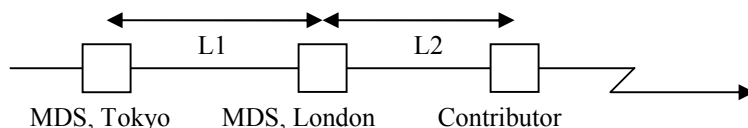
“Latency is our number one SLA”. So said one of JPA’s clients recently. But how do we measure that latency? And, more importantly, how can we make use of it to control how we trade? The answer lies in a suite of programs based around JPA’s **Tracker** product.

Sources of Latency

The central concept in Tracker is that of a **journey**. A journey is the path that a single item of data takes through a network. Each step along a journey is called a **datapoint**. As an example, consider a contribution to a vendor via Marketlink. Let’s complicate things slightly by saying that this item of data is published into market data system in Tokyo. However, the actual contribution is handled by a contribution process situated in London. So we have three datapoints:

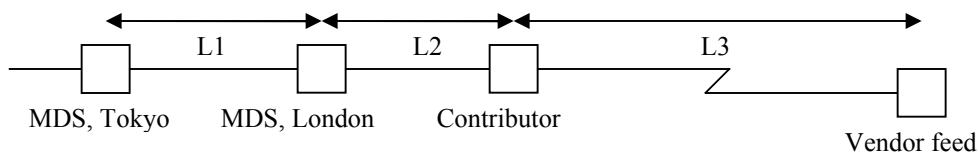
- MDS in Tokyo
- MDS in London
- Contribution server in London.

This is what this journey looks like:



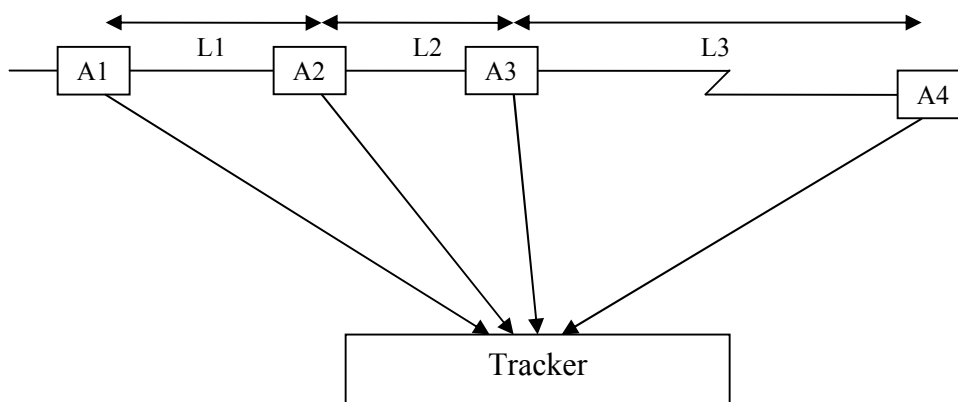
In this journey, our network has introduced two latencies. By measuring these, we can determine how well our network is performing, and take steps to improve things if the delays are significant.

However, all we’ve measured here is internal latency. We haven’t yet taken into account how our vendor is performing. And more often than not, that’s the biggest delay of all. So we need to add a fourth datapoint to our example – what our item looks like on the vendor’s datafeed:



Measuring Latency

So how does Tracker measure this latency? First of all, we need an **agent** at each datapoint, to watch the data in real time as it passes through. Secondly, we need a central monitoring process to receive the updates from each agent and collate them into a coherent sequence. This is what our monitoring network looks like:



Fortunately, because Tracker is based on the JPA **PlatformOne**TM suite of market data products, we already have a considerable set of agents to choose from. We have interfaces to all the major market data systems, as well as a wide range of datafeeds and other systems. We also have some slightly more obscure agents available, such as the one that monitors published SASS3 packets *before* they get re-distributed. And it goes without saying that any existing custom system can be turned into an agent very easily by using one of the extensive range of **PlatformOne**TM APIs. In addition to this, JPA's own range of contribution products all have agent capability built into them.

Agents can, incidentally, run in failsafe redundant mode, in which case the requests from Tracker are load-balanced between them, with failover capability in the case of failure. Tracker and – to a large extent – all of its agents can run under either Windows or Solaris.

So, let's imagine that an update to a data item has been contributed in Tokyo. The agent in Tokyo, A1, sends this update to Tracker, which receives it at time T1. And so on. Here's the picture that Tracker is starting to build up:

Update from A1:	T1
Update from A2:	T2
Update from A3:	T3
Update from A4:	T4

However, that's not the whole picture, by any means. We know that there are inherent latencies in our network, regardless of anything that our market data system is introducing. Tracker gets around this by constantly polling each of its agents to calculate the underlying latency, which is then removed from the incoming figure:

Update from A1:	$T1 - \delta t_1$
Update from A2:	$T2 - \delta t_2$
Update from A3:	$T3 - \delta t_3$
Update from A4:	$T4 - \delta t_4$

Finally, it calculates the latencies:

$$L1 = (T2 - \delta t_2) - (T1 - \delta t_1)$$
$$L2 = (T3 - \delta t_3) - (T2 - \delta t_2)$$
$$L3 = (T4 - \delta t_4) - (T3 - \delta t_3)$$

Taking Action

Tracker can be set up simply to log this information to a file for later analysis, typically for use in negotiations with the vendor in question. However, it can also take a more active role in proceedings. Because Tracker is based on **PlatformOne™**, which is itself a fully-fledged market data system, it also has the capability of generating its own real-time feed, containing latency information for each of the items being tracked. There are a number of things that can be done with this.

First of all, this feed can be read by a standard application in the Tracker suite, called **TrackMon**. This has a simple user interface, which shows details of all latencies, with the highest at the top, or just those that are outside a pre-set threshold. This can run on a completely different machine from the central Tracker server.

If, however, we need to integrate the output of Tracker into our existing monitoring system, then we could consider using the **PlatformOne™** SNMP server, which can be used to generate SNMP traps if the latency gets critical. Or we could simply pipe the feed into an existing market data system, such as RMDS.

In fact, Tracker is even more flexible than that, because **PlatformOne™** has its own set of APIs which can be used to integrate this tracking capability into existing applications. For example, it could be used to control an electronic trading system that automatically withdrew from the market if the latency went above a certain threshold. Any of the

standard **PlatformOne™** APIs can be used to watch the Tracker feed, and this includes Java, COM and .NET.

Comparing Data Sources

It's not just contributed data that can be monitored using Tracker. This tool has also been used to compare the same data from – for example – Reuters and Eurex, to establish whether or not it is better to take data direct from Eurex or via Reuters.

Monitoring and Fixing Data Quality

Checking Up

Minimizing the latency of the data is of no use whatsoever if the data itself is of poor quality, and that's where the **PlatformOne™ DataQA** product comes in. The purpose of DataQA is to monitor the outgoing data, watching out for discrepancies, and gathering general statistics. The central concept is that of a **checklist**. Each checklist can be applied to either a set of records or an entire dataset.

The following checks may be included in a checklist:

- Calculate average spread
- Calculate high and low
- Calculate maximum and minimum
- Calculate average update rate
- Check spread (against either an absolute or percentage tolerance, either fixed or taken from another field)
- Watch for idle (if no change is made to a rate for a pre-specified length of time)
- Watch for empty updates (updates with no changes to any of a specified set of fields)
- Log every tick

These checks may be applied at regular intervals (say, every 5 minutes) and/or over the whole of the working day.

Cleaning Up

Of course, a lot of problems with the quality of data can be fixed at source before they become a potential problem. And that's where the **PlatformOne™ Cleaner** product comes into its own.

Cleaner is a rule-based application that literally cleans the data in a feed. For example, it can cut out values that aren't within a preset range. If a value is too different from the

[JPA White Paper: Monitoring Market Data Latency and Quality](#)

[JPA/P1/WP/Tracker/1.1](#)

[Jonathan Pinnock, JPA, 26/9/06](#)

previous value, it can also be ignored. But it's also smart enough to realise that if a rate goes in an unexpected direction and keeps going in that direction, it's probably good after all.

The action taken when a value seems to be wrong can vary from holding back just that value, or holding back the whole update for the entire record. Alternatively, the update can be let through, but with a designated confidence field set to a lower value. This confidence value can also be set to drop if the record isn't updated in a while.

Cleaner doesn't stop at simply blocking data that it doesn't like the look of. It can also fix it automatically. For example, if a record contains a bid value that is bigger than the corresponding ask, it can be set to simply swap the two values around.

In Conclusion ...

In the end, it's all about peace of mind. It's almost impossible to underestimate the importance of timely, high-quality data. The **PlatformOne**[™] suite of products will give you the ability to know when there's a problem brewing long before your users and customers are affected by it, and the information you need to track down the root cause. It might even fix the problem before you get to hear about it in the first place.