Calc Farm – User Guide

Contents

[Architecture 2](#_Toc391579364)

[Start Calc Farm? 3](#_Toc391579365)

[Code Architecture 6](#_Toc391579366)

[How do we know that it works? 7](#_Toc391579367)

# Different Load Scenario, Different Tools

For example, PriceServer feeding in thousands of ticks per second. CalcServers sitting in the back picks up the tick, run some **lengthy** calculations (Quant Trading algo for instance). This is the most demanding scenario among the four cases and requires massive CalcServer farm to handle to information flow.

For example, PriceServer feeding in thousands of ticks per second. CalcServers sitting in the back picks up the tick, run some **simple/short** calculations such as pnl updates, simple aggregation…etc.

To minimize latency from PriceServer to CalcServer to trading screens, one may employ a message bus (Tibco, RabbitMQ…etc) as central backbone, connecting the various components.

**CASE 3.**

High Freq Short Calc

**CASE 4.**

High Freq Lengthy Calc

**CASE 1.**

Low Freq Short Calc

**CASE 2.**

Low Freq Lengthy Calc

For example, dayend/monthend mark-to-market, pnl, risk, stressing, reporting…etc.

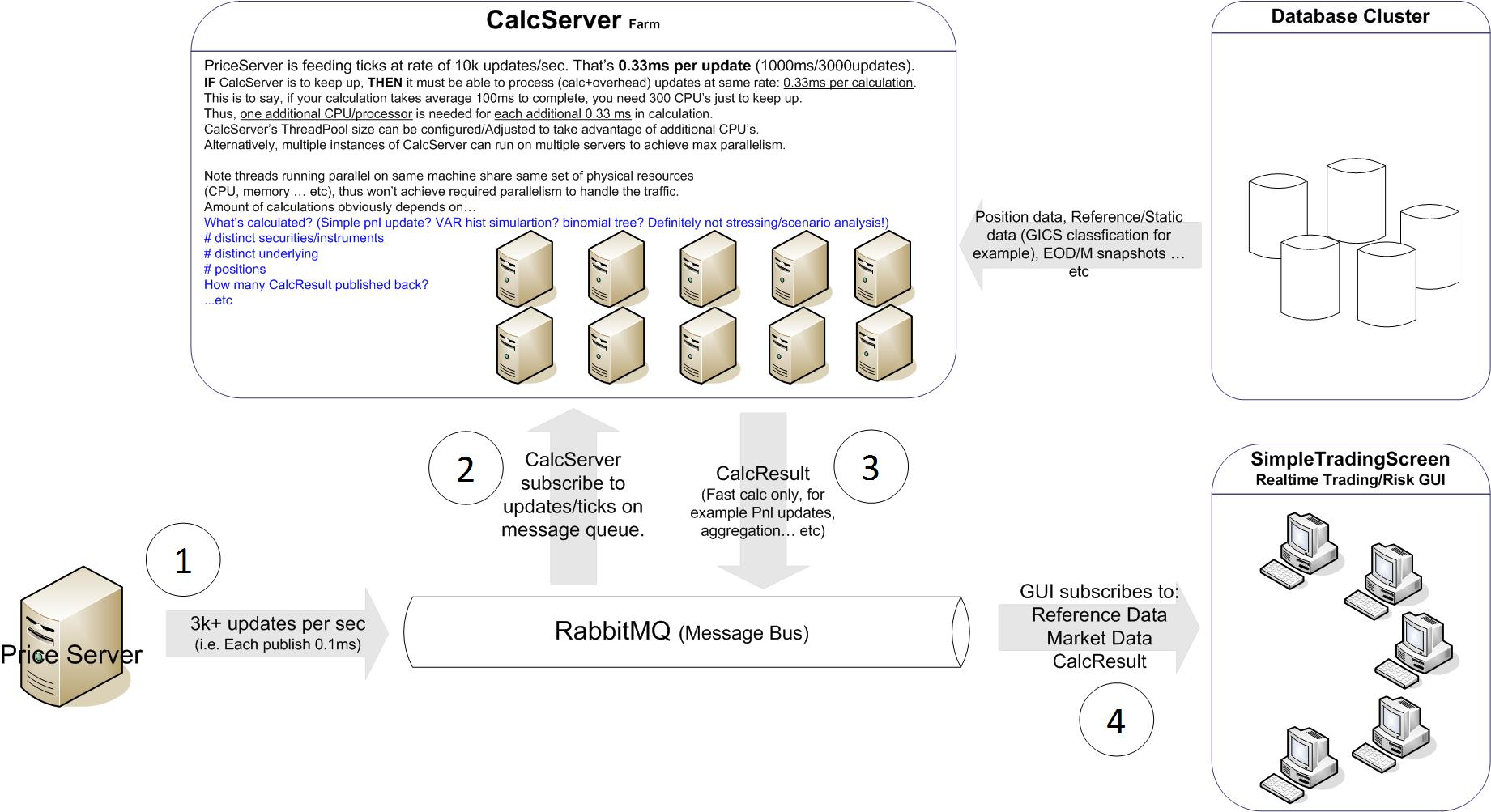
May employ Schedulers with load distribution grids support such as BMC Control-m, Appliedalgo.com or Autosys.

These products tend to have higher execution overhead (scheduling, caching…etc) so they are more suitable for grid applications hosting lengthy calculations (Calc duration 1 sec +)

Update Freq

Duration of Calculations

# Architecture



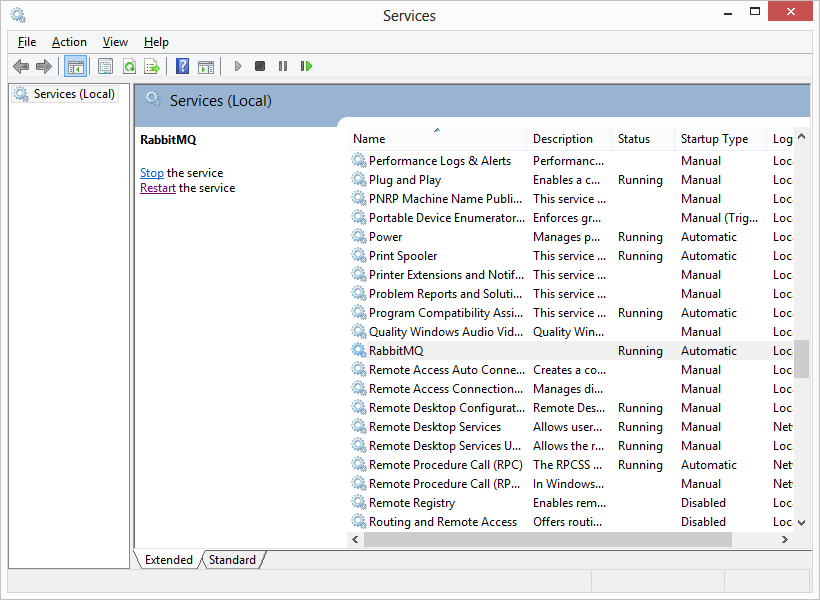
Create dummy data: Stock prices for HIS constituents.

**Remark:** Separate Visio diagram bundled with the package.

# Start Calc Farm?

**STEP 1.** RabbitMQ

Please download+install RabbitMQ, make sure the service is running.

****

**STEP 2.** Start CalcServer

From command prompt, navigate to \CalcServer\bin\Release\

Simply type **CalcServer**. There’re only three settings in app.config:

<appSettings>

<add key="QueueUrl" value="localhost"/> 🡪 Point at RabbitMQ

<add key="DetailLog" value="false"/> 🡪 For debugging only. If DetailLog=true, every price tick received would be logged and printed to screen. Set to false otherwise it will slow down CalcServer significantly.

<add key="MaxThreadPoolSize" value="8" /> 🡪 Maximum size of thread pool. Note, threads and concurrent instances of CalcServer on same physical machine share same set of Physical Resources, configure this wisely.

</appSettings>

**STEP 3.** Start PriceServer

From command prompt, navigate to \PriceServer\bin\Release\

Simply type **PriceServer**. There’re only four settings in app.config:

<appSettings>

<add key="QueueUrl" value="localhost"/>

<add key="DetailLog" value="false"/>

<add key="PerSecPublishThrottle" value="3000"/> 🡪 Max publish rate

<add key="MaxCountPublishesCulmulative" value="0"/> 🡪 Max cumulative # publishes before stop

</appSettings>

**STEP 4.** Start SimpleTradingScreen

From command prompt, navigate to \SimpleTradingScreen\bin\Release\

Simply type **SimpleTradingScreen**. There’re only two settings in app.config, I’m not going to repeat what’s already been covered.

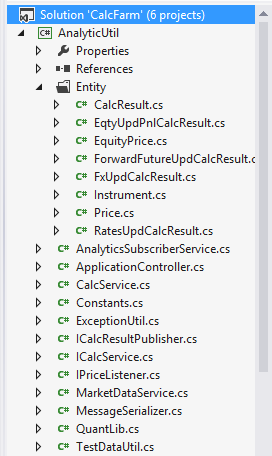
<appSettings>

<add key="QueueUrl" value="localhost"/>

<add key="DetailLog" value="false"/>

</appSettings>

# Code Architecture

The sample application is …

* Coded in .NET 4.5, C#
* Third Party Library used:
  + RabbitMQ – Message Bus

V2.4.1.0

<http://www.rabbitmq.com/>

* + Log4net – For logging

V1.2.13

<http://logging.apache.org/log4net/>

* Nunit – Unit testing, confirmation CalcServer not lagging behind PriceServer

V2.6.3

<http://www.nunit.org/>

* Components
* PriceServer: Feed Dummy price (HSI constituents) to message bus. Default throttled at 3000 updates per sec.
* CalcServer: Picks up price updates from message bus, run some dummy calculations, then publish back CalcResult to message bus.
* SimpleTradingScreen: Picks up CalcResult’s from message bus, and simply display to screen.
* AnalyticUtil – PriceServer, CalcServer, SimpleTradingScreen all reference this dll.

Price, CalcResult and their subclasses are under folder “Entity”.

Price and CalcResult are both decorated “abstract”. Subclasses are constructed for different asset classes and calculation types.

# How do we know that it works?

* Tested on Intel 2.6GHz (Single Processor) with 4GB RAM
* CalcServer dump performance statistics into “CalcServerStatistics”.

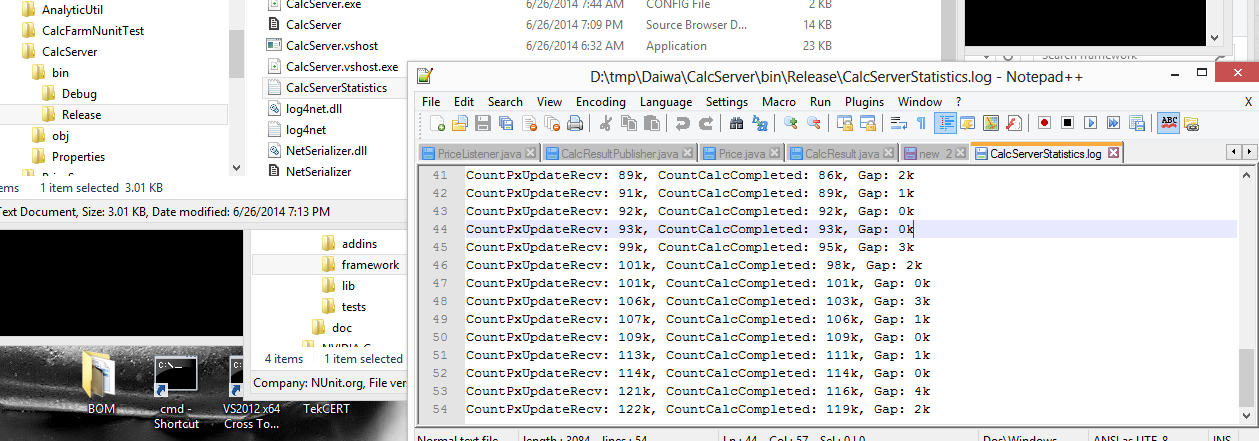
Basically,

* 1. **CountPxUpdateRecv** is # ticks CalcServer has picked up from RabbitMQ.
  2. **CountCalcCompleted** is # calculations (dequeue+calc+publish result) been completed.
  3. **Gap** = **CountPxUpdateRecv – CountCalcCompleted**

If CalcServer is able to keep in pace with PriceServer, what you’d see is Gap stabilize.

The following result is done with PerSecPublishThrottle = 3000 (i.e. 3000 ticks per second)

Try set this to, for example, 10000. With Single instance of CalcServer running you will start seeing ***“Gap Widening”*** (i.e. Gap keeps increasing most of the time)

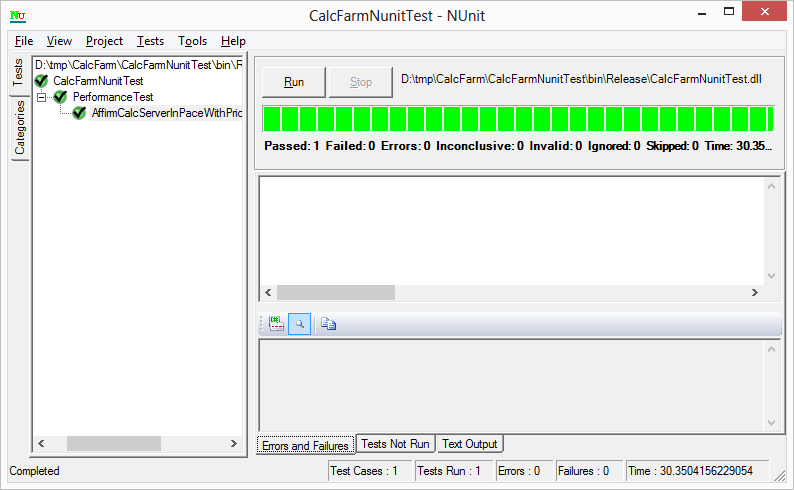


* Nunit

It basically kick starts CalcServer and PriceServer. Then examine CalcServer performance dump file \CalcServer\bin\Release\CalcServerStatistics.log.

Then assert that “Average Gap” < 10k

It’s same as manually examining " CalcServerStatistics.log”, and confirm that “Gap” isn’t monotonically increasing, which is an indication that CalcServer can’t keep up with PriceServer



* Youtube video: <http://youtu.be/grQ3fPOqeo4>