3Forge

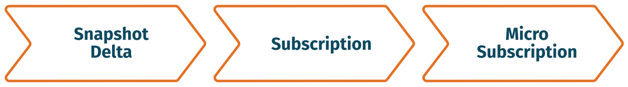
An Embedded Micro-Subscription Model

Case Study Series | Part 1

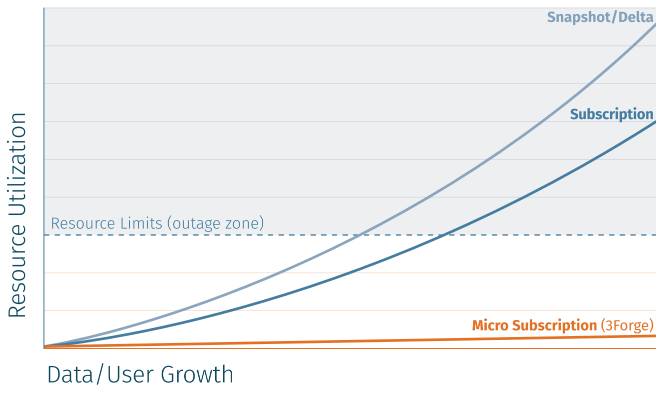
This is the first in a series of discussion papers outlining the core design challenges 3Forge has addressed. Our goal is to help clients transition from costly heavy front-end desktops to high performance, no load browser-based desktops, allowing participants to improve performance while reducing overall cost.

The problem is simple, backend systems generate lots of real-time data that many users' need instant access to. Doing this efficiently increases overall throughput and reliability while keeping costs low.

Throughout the years, there have been 3 major evolutionary phases, each building on the previous model to bring a more efficient use of resources.



When measuring performance, the focus is on how resource requirements grow as more users and data are added. This is important because a non-linear function will ever-more quickly approach and exceed resource limits resulting in platform outages.



And as organizations hit resource limits, they are forced to take costly actions which only provide temporary, incremental relief:

- Installation of extraordinary network infrastructures
- Top of the line, costly, desktop(s)
- Investments in expensive appserver farms.
- Investment in front-office staff, maintaining fragile desktop infrastructure

In the Beginning There Was Snapshot/Delta

Engineers created the "snapshot/delta" model with a centralized server process (appserver) that maintains all data. Each user has a desktop application that connects to the appserver and sends a request:

```
"Hey Appserver, give me a copy
of all your data"
```

The appserver responds with a full copy of the data (snapshot) supplemented with continuous updates (delta) so that the application can maintain its own updated dataset.

Outcome:

As many users connect, all data must be sent in duplicate from the appserver to each user. This means that as Data and/or users grow: (1) network bandwidth requirements increase, (2) desktop memory/processing needs grow, (3) the appserver's workload increases.

The nightmare scenario: Too much data being pushed to too many users causes network/application saturation. Slow consumers are disconnected and when they reconnect a new snapshot must be sent again, causing further resource shortages. End result is usually a costly systemic outage.

Next Came the Subscription Model

Instead of sending all data to all users all the time, the subscription model provides a granular approach so applications receive snapshot/deltas of only the data that a user might be interested in, ex:

```
"Hey Appserver, give me data for accounts
ACT123 and ACT456"
```

Outcome:

This has tangible benefits because the appserver and desktops are now communicating less data. But simple subscriptions still cast too wide of a net. In the sample subscription above, the resource overhead is based on how many records there are for accounts ACT123 and ACT456. And if those accounts are big, there are probably more people interested in monitoring them, so we're still left with an exponential growth issue: More users want to monitor more data.

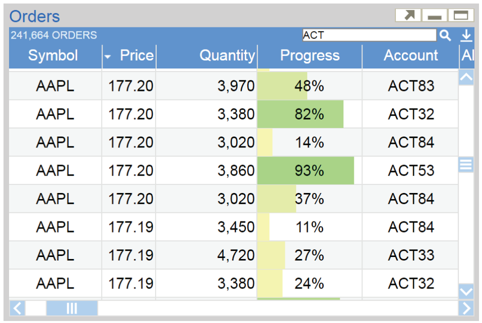
On top of that, the Pareto Principle suggests 80% of business comes from 20% of accounts (80/20 rule). This results in uneven data distribution making the subscription model far less effective than predicted.

Micro-Subscription Solves Exponential Growth Issue

The more specific a subscription is the more resource efficient it is. So in fact, the most efficient subscription is to only receive data for what the user can physically see on their screen(s). This is considered a Micro-Subscription, and we'll use 3Forge's AMI as an example as it fully embraces the feature set.

Requirement 1 - Fixed Size Subscription:

The 3Forge client will optimize the query based on the user's specific needs and generate as narrow as possible of a subscription. For example, consider the below visualization:



Symbol	Price	Quantity	Progress	Account
AAPL	177.20	3,970	48%	ACT83
AAPL	177.20	3,380	82%	ACT32
AAPL	177.20	3,020	14%	ACT84
AAPL	177.20	3,860	93%	ACT53
AAPL	177.20	3,020	37%	ACT84
AAPL	177.19	3,450	11%	ACT84
AAPL	177.19	4,720	27%	ACT33
AAPL	177.19	3,380	24%	ACT32

AMI will generate a micro-subscription that only asks for visible rows and columns based on custom filters, sorting, etc.:

```
"Hey Appserver, show me Sym, Px, Quantity,
Progress, Account for Orders sorted by Price
on rows 123,105 to 123,113 containing ACT"
```

Requirement 2 - Subscription Modification:

With the example above, let's say the user scrolls to the right, changing visible columns:

```
"Hey appserver, update the subscription to
show Quantity, Progress, Account, Alerts, Time"
```

Moving scroll bars, changing sorting, applying custom filters, adjusting window size, etc. cause AMI to instantly edit the subscription behind the scenes. The web server has been designed to support very fast modifications of a subscription instead of creating new ones each time.

Requirement 3 - Dynamic Data Rate Throttling:

Usually, the client only requires a consistent view and is not concerned with micro-fluctuations of data. Consider the scenario where a price jumps from 50.05 to 50.06 to 50.07 in under a millisecond. The 50.06 price is so brief it could be conflated such that the user just sees an update from 50.05 to 50.07. So a micro-subscription with conflation would look something like:

```
"Hey appserver, send me updates at most
every 10 milliseconds"
```

This guarantees the web browser will only receive a maximum amount of data, regardless of data rates coming in from the backend.

Dynamic data rate throttling can be effectively used to maintain reliability during usage bursts. For example, if suddenly 3x the number of users were to log on, then the webserver can recognize resource contention and dynamically adjust refresh rates to the browsers.

Outcome:

With these changes we've moved to a resource growth model that is independent of data size and rate, effectively going from an exponential growth model to a linear one that is a fixed cost based on the number of end users.

Here is a sample table showing the total bandwidth required to supply updates for a given number of records and users:

# of Records	# Of Users	Snapshot/ Delta*	Subscription*	Micro- Subscription*
500,000	50	23.8 MB/SEC	6 MB/SEC	**1.4 MB/SEC**
1,000,000	100	95.3 MB/SEC	23.9 MB/SEC	**2.9 MB/SEC**
5,000,000	500	2.4 GB/SEC	596 MB/SEC	**14.3 MB/SEC**
10,000,000	1000	9.5 GB/SEC	2.4 GB/SEC	**29 MB/SEC**

** Based of 100 fields per record. 1% of Fields updated every 10 seconds. Updates cost 10 bytes. Each user has 5,000 cells visible and scrolls every 2 seconds.*

Final Thoughts

3Forge set out to create a real-time user experience with minimum desktop load, accessible in a light-weight browser because it provides many advantages over a heavy-weight desktop application. But the browsers limited processing and memory abilities forced us to design an ultra-efficient subscription pattern and the result has been a web-based user interface with a natural look and feel, incredible performance and scalability. Today, over 20% of equities orders are being monitored through our web interface on a daily basis and the low-overhead micro-subscription model has provided unmatched reliability (99.99%) and scalability indifferent of industry or region.

Subscription topics that are supported by 3Forge, but out of scope for this paper include:

- Horizontal Scaling of appservers
- Subscriptions to aggregation/enriched data

For more information, email info@3Forge.com.