Views v0.3 / pi v0.2

The basic advice regarding response times has been about the same for thirty years [Miller 1968; Card et al. 1991]:

0.1 second is about the limit for having the user feel that the system is reacting instantaneously, meaning that no special feedback is necessary except to display the result.

1.0 second is about the limit for the user's flow of thought to stay uninterrupted, even though the user will notice the delay. Normally, no special feedback is necessary during delays of more than 0.1 but less than 1.0 second, but the user does lose the feeling of operating directly on the data.

10 seconds is about the limit for keeping the user's attention focused on the dialogue. For longer delays, users will want to perform other tasks while waiting for the computer to finish, so they should be given feedback indicating when the computer expects to be done. Feedback during the delay is especially important if the response time is likely to be highly variable, since users will then not know what to expect.

Need for Speed

Too slow:

- Node.js
- MongoDB
- qPid, AMQ

- HighCharts.js
- jQuery

Need for Speed

Fast enough:

- Varnish
- Redis
- MySQL
- ZeroMQ

raphaël.js

GreenSock Animation Platform

HTML5 native functions

CSS3 transformations

Need for Speed

- Aggregér på øverste nivå under innsamling av data
- Gjør batch-inserts i MySQL
- Bruk Redis som sentralt applikasjonsminne, bruk unix sockets til kommunikasjon, og igbinary som serializer
- Bruk Redis til PHP-sesjoner
- Best-practice GIF beacon

Need for Speed

- AJAX: PING + HTTP-overhead (ca. 120 ms)
- WebSocket: PING + TCP-overhead (<40 ms)
- PING-tid innenlands i Norge ligger på 20-100ms

- Modulær innlasting over WebSocket
- Caching av js-/css-moduler i LocalStorage
- Bakgrunnsprosessering i WebWorker

Need for Speed

MySQL v Redis

- MySQL query: a few milliseconds
- Redis cache check: a few microseconds

Out with The Old, in with The New

OUT

- AJAX
- XML
- MVC/MVV
- data binding
- callback pasta
- Node.js
- NoSQL
- Jquery

IN

- WebSockets
- JSON
- Flow-Based Programming
- Self-contained components
- WebWorkers & messaging
- Redis pubsub
- MySQL with table templates
- HTML5 native functions

A Brand Spanking New Idea: π

- Use message passing across a namespace to tie any number of clients and server services together.
- Open a bridge between the client and server messaging hubs, creating a namespace across the entire ecosystem, each with a single point of entry, making User Access Control a breeze.
- Add services to the namespace by writing scripts and programs that publish their progress and status within their own namespaces.
- Servers written in PHP have access to the current PHP session even when running as CLI.

A Brand Spanking New Idea: *π*

pi is a simple, lightweight, flexible and fast application platform for HTML5 compatible browsers.

It features:

- modular loading and dependency injection of JS and CSS
- session support
- utility libraries
- a set of conventions for how to implement an interface
- a visual component library called *pcl*
- a *pcl* charting library using SVG and raphaël.js
- streaming live JSON data packets over SSE (Server-Sent Events) / EventSource

```
-   <head>
-       <script>
-         var
-           π  = π  || {}, pi = pi || π;
-
-         // activate the debugger
-         π.DEBUG = true;
-
-         //make a note of when we started up.
-         pi.__sessionstart = (new Date()).getTime();
-       </script>
-
-   ...
-
-
-     <!--  bootstrap  -->
-     <script src="../../assets/js/pi.js"></script>
-
-   </body>
-
```

```
-    var
-        π  = π  || {},
-        pi = pi || π;
-
-
-    /*  ----  Our top level namespaces  ----  */
-      π.events      = π.events      || { _loaded: false, _ns: 'events' };
-      π.srv         = π.srv         || { _loaded: false, _ns: 'srv' };
-      π.app         = π.app         || { _loaded: false, _ns: 'app' };
-      π.pcl         = π.pcl         || { _loaded: false, _ns: 'pcl' };
-      π.session     = π.session     || { _loaded: false, _ns: 'session' };
-      π.system      = π.system      || { _loaded: false, _ns: 'system' };
-      π.debug       = π.debug       || { _loaded: false, _ns: 'debug' };
-      π.io          = π.io          || { _loaded: false, _ns: 'io' };
-
-      π.util        = π.util        || { _loaded: false, _ns: 'util' };
-      π.math        = π.math        || { _loaded: false, _ns: 'math' };
-      π.statistics  = π.statistics  || { _loaded: false, _ns: 'statistics' };
-
-      // your plugins here, like so:   pi.plugins.yourcompany.yourplugin.[whatever] = { # your plugin object };
-      π.plugins     = π.plugins     || { _loaded: false, _ns: 'plugins' };
-
-      // for all you crazy cowboys, your own playground
-      π.maverick    = π.maverick    || { _loaded: false, _ns: 'maverick' };
-
-
-      π.PI_ROOT    = "assets/js/";
-      π.LIB_ROOT    = "../../assets/js/";
-      π.SRV_ROOT    = "../../../srv/";
-
```

```
-       π.inject = function (src, elem) {
-         var
-           element   = elem || document.body,
-           fragment  = document.createDocumentFragment(),
-           container = document.createElement("div");
-
-         container.innerHTML = src;
-         fragment.appendChild(container);
-         element.appendChild(fragment);
-       };
-
-
-       π.require = function(module, async, defer, callback){
-
-         if (π.loaded[module.replace(/\./g,'_')]) {
-           if(callback) {
-             this.callback.call("loaded");
-           }
-           return true;
-         }
-
-         var
-           cursor  = document.getElementsByTagName ("head")[0] || document.documentElement,
-           path    = '../../assets/js/pi.',
-           script  = document.createElement('script');
-
```

GIF beacon done Right

Method works in any browser

- Encode event object in GIF url
- Request the url with an Image object, do not touch the DOM
- Catch encoded events in varnish
- Return HTTP 204 No-Content
- Saves one HTTP packet
- Only 204 means success
- Pipe 204 events with varnishlog into an aggregator script or program
- Publish live aggregates on Redis pubsub channels
- Batch-insert collected data into MySQL at set intervals
- Keep all application data in Redis
- Good for collecting huge amounts of data from huge amounts of user sessions
- No return channel

WebService API

- JSON WebService
- With PHP session
- Good for collecting data from a large number of individual user sessions
- Too slow for application use

WebSocket API

- JSON messaging
- Full duplex connection
- With PHP session
- Extremely low latency (PING + microseconds)
- No gzip
- Only connection that is fast enough for application use

SSE/EventSource API

- JSON event stream
- With PHP session
- Extremely low latency (PING + microseconds)
- No gzip
- No return channel
- Good for streaming the same data to multiple clients

Bottlenecks

- Incoming: Probably varnish and/or NIC, but needs verification
- Outgoing: Each WebSocket session uses a separate port, limiting the maximum number of concurrent sessions to tens of thousands. Each session will also run as a separate thread on the server.

Scalability

- Incoming: Put HAProxy in front of several Varnish servers
- Incoming: Use Redis version with clustering support
- Outgoing: Implement a generic server that is able to handle multiple concurrent sessions in a single-thread event loop. Make the generic server transparent to the namespace, so multiple instances can run on the same server
- Application data sharing: Use multiple instances of single-threaded Redis servers
- Inherent: Use one virtual server for each application

Reliability

- Increase reliability: Exchange AMQP or similar for Redis pubsub => guaranteed message delivery, but at greater cost
- Varnish + LAMP = The Gold Standard of reliable, high-speed performance

What's unique?

- Namespacing
- Scoping
- Addressing

- Bridging
- Mapping
- Class/instance data storage

Namespacing

Use namespaces throughout the system for organization of files, classes, objects and session data:

- Pi installation
- App
- Component
- User

- Connect any two points throughout the system.
- Broadcast using wildcards
- Use point-to-point delivery

Scoping

Connect namespaces through routing across client/server bridge, in order to create four scopes:

- System (installation of *pi*)
- App (running on system)
- Component (running within app)
- User

- Rewrite addresses at bridge to create «subnets»
- Single point of access => easy access control
- Connect any two points throughout the system.

From Rewrite to Rewiring

Use rule-based routing to connect to server-side services and storage:

- Rewrite addresses at bridge to create «subnets»
- Rewrite addresses to store data at correct level of specificity: company, application, user, session, component, instance

Through The Looking Glass

- You cross the cli/srv bridge and where you end up, depends on where you came from.

Take *pcl*

- Component config and data is stored under:

  pi.srv.pcl.[company].[component]
- Component config within app is stored at:

  pi.srv.pcl.[company].[app].[component] This is for *all* app users.
- Component data storage at instance level:

  pi.srv.pcl.[company].[app].[company].[id]

- Connect *pi.srv.sessions.[app.company.user]* server-side to *pi.app.session* client-side, and create «data spaces» transparent to client app code, through routing at the client/server bridge

Go Global

π lives in the global scope of the browser's JavaScript VM. A global app object is insanely useful in combatting «wrong scope» fatigue and callback pasta.

- Any part of pi can always create a pointer to itself, and knows it's local and systemic address.
- Our client session maps directly to the server session.

The Power of Conventional Thinking

- Use **conventions** to overcome problems arising from loose typing and anonymous scoping
- Don't (just don't) try to create a self-describing data interchange format (looking at you, **XML**)
- **Trusted code** always obeys conventions
- Some words are **reserved** by convention
- **π.** is for sysdevs only, everybody else uses **pi.**
- Any code starting with *π.* is **part** of the platform
- Any code starting with *pi.* is **using** or **extending** the platform

Protect Your Privacy

- Objects whose name starts with a single underscore, are considered **_protected**.
- Objects whose name starts with a double underscore, are considered **__private**.
- **_protected** functions/variables should only be accessed by trusted code.
- **__private** functions/variables should only be accessed by the object itself

Philosophy

- Just-in-time (JIT)
- Batch processing
- Background workhorses
- Speculative execution
- Self-contained units
- Pass-through data routes
- Follow conventions
- Use best practices

The Future

- Go binary (JS typed arrays)
- Go compiled or LLVM (C / Pascal / Python)

The End Station

Go point-to-point binary with

ZeroMQ*

* just add namespace

Advantages of HTML5

- Works on devices
- Works on Windows Metro (?)
- Works on Adobe AIR
- Works in any modern browser
- Works on Smart-TVs