SCALING ERLANG WEB APPLICATIONS 100 to 100K users at one web server

Fernando Benavides (@elbrujohalcon)

Inaka Labs

March 23, 2012





HELLO WORLD!

- I'm a developer since I was 10
- I'm an Erlang developer since 2008







HELLO WORLD!

- I'm a developer since I was 10
- I'm an Erlang developer since 2008







Hello World!

- I've built several dynamic web servers
 - Many of them with real-time updates
 - Most of them with high scale requirements
- I'll show you how I make them scale





HELLO WORLD!

- I've built several dynamic web servers
 - Many of them with real-time updates
 - Most of them with high scale requirements
- I'll show you how I make them scale





HELLO WORLD!

- I've built several dynamic web servers
 - Many of them with real-time updates
 - Most of them with high scale requirements
 - I'll show you how I make them scale





Hello World!

- I've built several dynamic web servers
 - Many of them with real-time updates
 - Most of them with high scale requirements
- I'll show you how I make them scale





We will work on the scalability of a *web* project that has an *HTTP API* and a component that keeps clients *connected* to the server for *long periods* of time.

Examples:





We will work on the scalability of a *web* project that has an *HTTP API* and a component that keeps clients *connected* to the server for *long periods* of time.

Examples:





We will work on the scalability of a *web* project that has an *HTTP API* and a component that keeps clients *connected* to the server for *long periods* of time.

Social sites





We will work on the scalability of a *web* project that has an *HTTP API* and a component that keeps clients *connected* to the server for *long periods* of time.

Examples:

Social sites

Chat sites

Sports sites



We will work on the scalability of a *web* project that has an *HTTP API* and a component that keeps clients *connected* to the server for *long periods* of time.

Examples:

- Social sites
- Chat sites
- Sports sites





We will work on the scalability of a *web* project that has an *HTTP API* and a component that keeps clients *connected* to the server for *long periods* of time.

Examples:

- Social sites
- Chat sites
- Sports sites





We will work on the scalability of a *web* project that has an *HTTP API* and a component that keeps clients *connected* to the server for *long periods* of time.

Examples:

- Social sites
- Chat sites
- Sports sites





SCOPE

We will try to improve the way we use

- OTP behaviours
- TCP and HTTP connections
- Underlaying system configurations

We will not deal with

- Multiple machines/nodes
- Database choices and/or implementations





SCOPE

We will try to improve the way we use

- OTP behaviours
- TCP and HTTP connections
- Underlaying system configurations

We will not deal with

- Multiple machines/nodes
- Database choices and/or implementations





A soccer match is played at some stadium







Soccer fans are connected to the internet in their offices







A reporter is at the stadium with his device

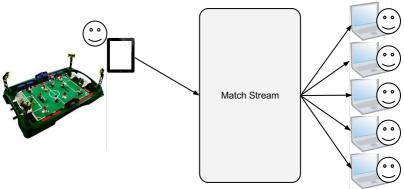








MATCHSTREAM connects them in real time







SYSTEM CHALLENGES

- Tons of concurrent users
- Two-hour-long bursts of connections followed by long periods of inactivity
- Real-time updates





SYSTEM CHALLENGES

- Tons of concurrent users
- Two-hour-long bursts of connections followed by long periods of inactivity
- Real-time updates





SYSTEM CHALLENGES

- Tons of concurrent users
- Two-hour-long bursts of connections followed by long periods of inactivity
- Real-time updates

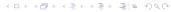




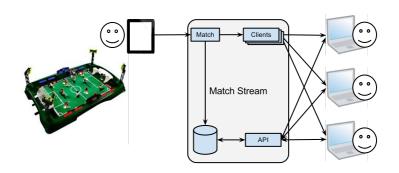
SYSTEM CHALLENGES

- Tons of concurrent users
- Two-hour-long bursts of connections followed by long periods of inactivity
- Real-time updates



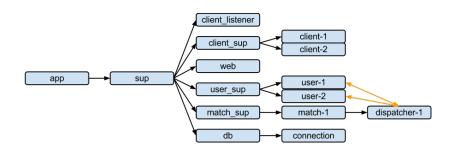


MATCH STREAM GENERAL DESIGN



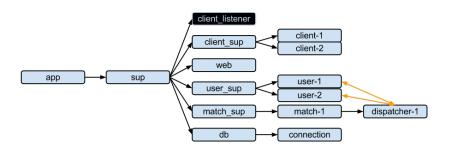








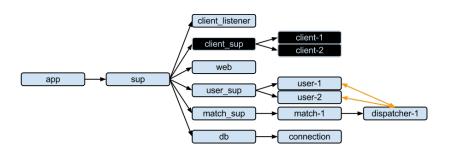




CLIENT_LISTENER gen_server. Listens on a TCP port to receive client connections

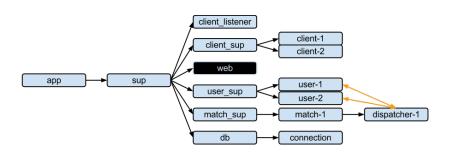






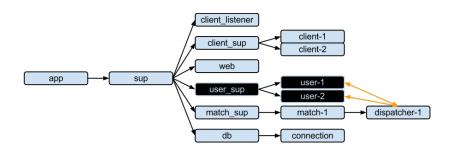
CLIENT_SUP supervisor. Supervises connection processes CLIENT gen_fsm. Handles a TCP connection





WEB mochiweb server. Listens for HTTP API calls

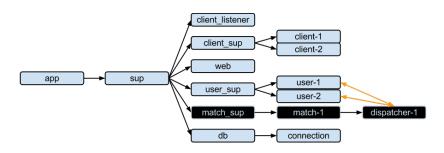




 ${\tt USER_SUP} \ \ {\tt supervisor}. \ \ {\tt Supervises} \ \ {\tt user} \ \ {\tt processes}$

user gen_server. Subscribes to match dispatchers and sends events to clients





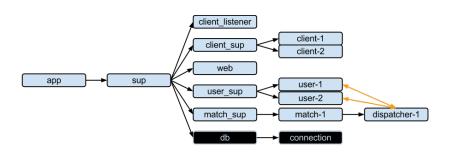
MATCH_SUP supervisor. Supervises match processes

MATCH gen_server. Listens to match events, stores them

DISPATCHER gen_event dispatcher. Delivers match events







DB gen_server. Processes database operations
CONNECTION erldis client. Handles the connection to the
database





LESSON LEARNED

Using Erlang to build your system is **not enough** to ensure **scalability**





Stage 1: The Original System Stage 2: OS Tuning Stage 3: Erlang Tuning Stage 4: Multi-Node Tuning

STAGE 1

TESTING THE SYSTEM AS IT IS

GOALS

• Find how much the system can handle

STEPS

- Create automated testers
- Start the system on a clean machine
- Test repeatedly adjusting the number of connections
- Have a human trying the system himself



Stage 1: The Original System Stage 2: OS Tuning Stage 3: Erlang Tuning Stage 4: Multi-Node Tuning

STAGE 1

TESTING THE SYSTEM AS IT IS

GOALS

Find how much the system can handle

STEPS

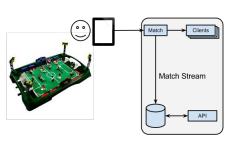
- Create automated testers
- Start the system on a clean machine
- Test repeatedly adjusting the number of connections
- Have a human trying the system himself





Introduction Match Stream Scaling Final Words Stage 1: The Original System Stage 2: OS Tuning Stage 3: Erlang Tuning Stage 4: Multi-Node Tuning

STAGE 1 RESULTS











- 1000 users
- 25 at a time
- 26s ART





STAGE 2

IMPROVING THE ENVIRONMENT

GOALS

• Improve the system environment without altering the code

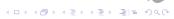
SETTINGS TO TUNE UP

Concurrent TCP connections

Open files limit

TCP backlog size





Stage 2

IMPROVING THE ENVIRONMENT

GOALS

• Improve the system environment without altering the code

- Concurrent TCP connections
- Open files limit
- TCP backlog size
- TCP memory allocation
- Erlang VM startup parameters





STAGE 2

IMPROVING THE ENVIRONMENT

GOALS

Improve the system environment without altering the code

- Concurrent TCP connections
- Open files limit
- TCP backlog size
- TCP memory allocation
- Erlang VM startup parameters





STAGE 2

IMPROVING THE ENVIRONMENT

GOALS

• Improve the system environment without altering the code

- Concurrent TCP connections
- Open files limit
- TCP backlog size
- TCP memory allocation
- Erlang VM startup parameters



STAGE 2

Improving the Environment

GOALS

• Improve the system environment without altering the code

- Concurrent TCP connections
- Open files limit
- TCP backlog size
- TCP memory allocation
- Erlang VM startup parameters



STAGE 2 Improving the Environment

GOALS

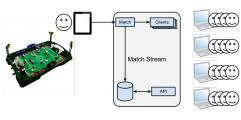
• Improve the system environment without altering the code

- Concurrent TCP connections
- Open files limit
- TCP backlog size
- TCP memory allocation
- Erlang VM startup parameters





STAGE 2 RESULTS



N = 4096 / C = 4

- 4096 users
- 4 at a time
- 9s ART





STAGE 3 IMPROVING MATCH STREAM

GOALS

Tune up the system for one node

STEPS

- Find a problem
- Fix it using the list of Tips and Tricks
- If not there, add it
- Repeat from Stage 1





STAGE 3 IMPROVING MATCH STREAM

GOALS

• Tune up the system for one node

STEPS

- Find a problem
- Fix it using the list of Tips and Tricks
- If not there, add it
- Repeat from Stage 1





STAGE 3.1 CONNECTION TWEAKS

BACKLOG

- Allow more concurrent connections
- Don't forget TCP tuning your HTTP server

CONNECTIONS

- Don't use just one of them
- Check inbound and outbound connections





STAGE 3.1 CONNECTION TWEAKS

BACKLOG

- Allow more concurrent connections
- Don't forget TCP tuning your HTTP server

Connections

- Don't use just one of them
- Check inbound and outbound connections





STAGE 3.1 Connection Tweaks

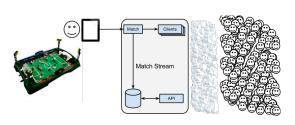
SYSTEM ARCHITECTURE







STAGE 3.1 RESULTS



N = 65536 / C = 8192

- 8192 users
- 256 at a time
- 16s ART





STAGE 3.2

SUP HANDLER

- Don't use it
- Monitor the processes instead

Long Delivery Queues

Use repeaters





STAGE 3.2

SUP HANDLER.

- Don't use it
- Monitor the processes instead

Long Delivery Queues

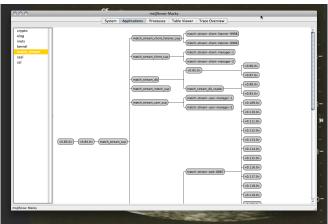
• Use repeaters





STAGE 3.2

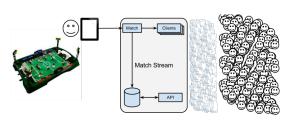
SYSTEM ARCHITECTURE







STAGE 3.2 RESULTS



N = 65536 / C = 8192

- **8192** users
- 256 at a time
- 8s ART





STAGE 3.3 GEN_SERVER

CALL TIMEOUTS

Remember gen_server:reply/2

MEMORY FOOTPRINT

Remember hibernate

LONG INIT/1

Use 0 timeout





STAGE 3.3 GEN_SERVER

CALL TIMEOUTS

Remember gen_server:reply/2

Memory Footprint

Remember hibernate

LONG INIT/1

Use 0 timeou





STAGE 3.3 GEN_SERVER

CALL TIMEOUTS

Remember gen_server:reply/2

Memory Footprint

Remember hibernate

LONG INIT/1

Use 0 timeout



STAGE 3.3 GEN_SERVER

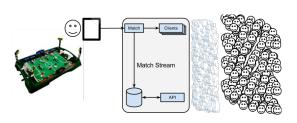
SYSTEM ARCHITECTURE







STAGE 3.3 RESULTS



N = 65536 / C = 8192

- 32768 users
- 1024 at a time
- 1s ART





STAGE 3.4 SUPERVISORS

- Sometimes simple_one_for_one supervisors get overburdened because they have too many children
- Try a supervisor hierarchy with several managers below the main supervisor
- Turn supervisor:start_child/2 calls into something like





STAGE 3.4 SUPERVISORS

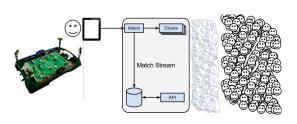
SYSTEM ARCHITECTURE







STAGE 3.4 RESULTS



N = 65536 / C = 8192

- 65536 users
- 2048 at a time
- 1s ART





TIMERS

- Don't use the timer module
- Use erlang:send_after

Logging

- Don't log too much
- Use a good logging system

REGISTRATION

- Sometimes it's better to register processes instead of keeping track of their pids manually
- You can always register processes both locally and globally





Timers

- Don't use the timer module
- Use erlang:send_after

Logging

- Don't log too much
- Use a good logging system

REGISTRATION

- Sometimes it's better to register processes instead of keeping track of their pids manually
- You can always register processes both locally and globally



TIMERS

- Don't use the timer module
- Use erlang:send_after

Logging

- Don't log too much
- Use a good logging system

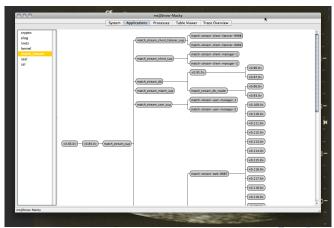
REGISTRATION

- Sometimes it's better to register processes instead of keeping track of their pids manually
- You can always register processes both locally and globally





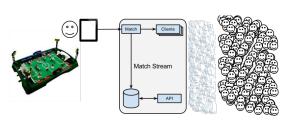
SYSTEM ARCHITECTURE







STAGE 3.5 RESULTS



N = 65536 / C = 8192

- 65536 users
- 8192 at a time
- 10ms ART





STAGE 4 Adding Nodes

GOALS

Find the best system topology

STEPS

- Prepare the system to run in more than one node
- Decide if nodes should be connected or independent
- Decide if nodes should be on the same machine or not





STAGE 4 Adding Nodes

GOALS

Find the best system topology

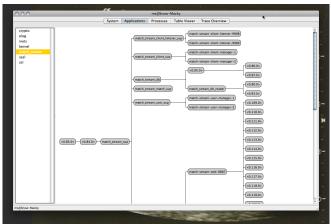
STEPS

- Prepare the system to run in more than one node
- Decide if nodes should be connected or independent
- Decide if nodes should be on the same machine or not



STAGE 4 Adding Nodes

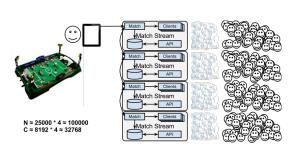
SYSTEM ARCHITECTURE







STAGE 4 RESULTS



- 100K users
- **32768** at a time
- 10ms ART





- This is an iterative process
- It worked awesomely for us in both experimental and real-life systems
- It's no silver bullet
- The list of Tips and Tricks grows constantly over time





- This is an iterative process
- It worked awesomely for us in both experimental and real-life systems
- It's no silver bullet
- The list of Tips and Tricks grows constantly over time





- This is an iterative process
- It worked awesomely for us in both experimental and real-life systems
- It's no silver bullet
- The list of Tips and Tricks grows constantly over time





- This is an iterative process
- It worked awesomely for us in both experimental and real-life systems
- It's no silver bullet
- The list of Tips and Tricks grows constantly over time





SCALING TOPICS

THAT WEREN'T COVERED ON THIS PRESENTATION

- Managing many nodes
- Choosing databases
- System specific improvements
- Measuring tools





QUESTIONS







Thanks!



