Erlang and First-Person Shooters

10s of millions of Call of Duty Black Ops fans loadtest Erlang

Malcolm Dowse
Demonware, Dublin
Overview

• History of Demonware
  – Who are we and what we do?
  – Why we switched to Erlang 4-5 years ago

• Our server-side architecture
  – How we use Erlang now

• What we have learned
  – Mistakes made
  – What we think would be great in the future
  – What we love about Erlang
Demonware – What we do

1. Multiplayer
   • Middleware for client-client game state transport
     • Encryption / NAT Traversal
     • Connection management
     • Peer-to-peer / Star topology
Demonware – What we do

2. Lobby servers
   - Matchmaking
   - Leaderboards
   - Stats Storage
   - Messaging/Chat
   - Audio/Video
   - Website Linking
   - Friends/Teams
   - Anti cheat
History

- Founded in 2003 in Dublin
  - Developing middleware for game studios
- In 2005..
  - Started hosting lobby servers
- In 2007..
  - Switched to using Erlang
  - Acquired by Activision (now Activision-Blizzard)
- In 2011..
  - One of the world’s largest online game service providers
  - 60+ employees, Dublin and Vancouver offices
Games that use us

Call of Duty
Games that use us

...and many more!
What we support

• The full online infrastructure for Call of Duty Black Ops
  – the world’s current best selling game.
• Four of the top 10 games on Xbox Live
• Over 2 million concurrent users
  – Comparable in size to Xbox Live
• Over 150 million registered users
• Cross platform:
  – Xbox 360, PS3, Wii, PC, iPhone/iPad
  – Coming soon: 3DS, PSP2
How we got into Erlang
The beginning..

• Mid 2003
  – Founded by former Trinity College Dublin students.
  – Aim: sell client-side networking middleware to games studios.

• Late 2004
  – Lots of polite interest; few customers.
  – Game studios wanted online servers, not middleware.

• Started creating a lobby services platform
  – Xbox 360 had Xbox Live. It set the standard.
  – Games studios needed something for Playstation (and PC)
2005 – C++/C++/Mysql

• Homebrew C++ server
  – Single-threaded
  – Dispatch requests into sub-processes per service
  – Application logic was in C++ and used Mysql

• Problems
  – One OS process per connected user is really bad
    • Max of 80 concurrent users
    • Luckily the first game didn’t sell well enough to hit that limit.
  – C++ crashes a lot if code is immature
    • Code was immature.
    • It crashed a lot.
2005/2006 – C++/Python/Mysql

- Rewrote all C++ business logic in Python
  - Maintained a pool of OS processes
- Kept core server in C++
  - Handles 1000s of concurrent connections
  - Encrypts, decrypts, dispatches requests
  - Asynchronous messaging between clients
  - Licenses and duplicate login detection
- Problems remain
  - C++ is the wrong language for concurrency
  - Code was becoming impossible to maintain
  - Poor error handling / debugging / metrics / scalability
  - Had to disconnect all users to change configuration.
2007 – Erlang/Python/Mysql

• Late 2006 / early 2007.
  – Former developer rewrote the C++ server in Erlang
  – Got a basic prototype running after a few weeks
  – ~4 months of development before used by games studios.
  – Went live for first time in mid-2007

• Improvements
  – Robust: didn’t crash.
  – Easier configuration
    • able to reconfigure everything without affecting clients
  – Better logging and administration tools
  – Faster to develop features, far fewer lines of code
Demonware in 2007

• Lots of customers
  – Activision, Ubisoft, Codemasters, THQ.
  – Acquired by Activision in May.

• Some big games..
  – Splinter Cell Double Agent, Saints Row, Worms Open Warfare, Colin McRae DiRT, Enemy Territory Quake Wars

• But no monster blockbuster
  – 20,000 concurrent users was a big title..

• Still a tiny company
  – 11 devs, 3 ops, 3 managers
Late 2007 – A blockbuster arrives
Late 2007 – A blockbuster arrives

- The most popular game on the (then new) PS3
- Much pain and suffering for us
  - .. and frustration for gamers.
  - Number of users grew continually for 5 months.
  - Every weekend brought a different bottleneck
  - Lots of outages and late nights
- It was a crisis for the company..
  - We had to grow up.
  - Erlang caused us relatively very few issues
  - Without the switch to Erlang the crisis could have been a disaster.
2007 and onwards

- **Continual growth**
  - In concurrent online users (20k to 2.5 million)
  - In requests per second (500 to 50k)
  - In servers (50 to 1850)
    - Spread across many data centres
  - In staff (17 to 60)
    - Spread evenly between Vancouver and Dublin
  - In competence!

- **And many new features/services**
  - The Black Ops launch (2010) was colossal
  - Many separate standalone components
  - Erlang/Python/Mysql is the core, but now with many exceptions
How we use Erlang
How we use Erlang

• Our core server for controlling Python
  – Managing 100,000s of concurrent TCP connections
  – Scheduling/queuing of tasks for python
  – Metrics gathering (SNMP)
  – Presence server (fragmented mnesia)
  – Message passing

• Other standalone game-related servers
  – Transient in-game data
  – Testing bandwidth
  – Ranking leaderboards

• In general:
  – for concurrency, and gluing sequential code together
TCP connections / task scheduling

• Two erlang processes per connected user
  – simple_one_for_one supervisor

• Delegate work to python OS processes
  – managed by a large supervision tree
  – dedicated task queues for some request types
  – Can restart/update python code without affecting users

• Periodic tasks
  – Use a modified timer module.
A presence server

• Needed to
  – Ensure a user can’t be logged in twice
  – Prevent duplicate license keys (PC)
  – Provide consistent, distributed snapshot of who is connected
  – In-game messaging

• Use fragmented mnesia
  – Scales linearly
  – Robust

• Our biggest single cluster:
  – 60+ 16-core Dell RC10s
Metrics / SNMP

- The erlang SNMP libraries get good use
- Vital for monitoring
  - online users
  - requests per second
  - request times
  - queue times
  - logins/logouts per second
  - disconnect reasons
- The workhorse is `ets:update_counter`.
- Easy to auto-generate cross-cluster metrics
Configuration

• Each game has a different, often complex configuration
• Our Erlang configuration code allows
  – Complex option settings and validation
  – Defaults, instantiation, inheritance
  – Cross-cluster upgrades
  – Rollback on failure
  – Language agnostic
  – Puppet integration
• Making something configurable should be simple and painless
Webconsole/webservices

• YAWS is used internally
  – Webconsole
    • Live debugging
    • Local development
  – Webservice interface
    • Games studios can remotely
      – Update the message of the day
      – See how popular certain game features are
    • Used by us to control to our clusters remotely
Game-related services

• Leaderboard ranking
  – Keeps huge leaderboards (15m+ users) ranked in real time.
  – Uses ETS and a modified gb_trees module.
  – The rank is a feature of the tree itself

• In-memory key-value store
  – Built on ETS.
  – Grouping online users into categories
  – Dynamic chat channels
  – Presence information

• Bandwidth testing
  – UDP packet blast against an erlang server
  – Client gets an estimate of his bandwidth.
Some Lessons we’ve Learned about Erlang
Lessons: Basics, but important

• Learn to use the core datatypes:
  – Iolists, records (not tuples), binaries/bitstrings, refs, atoms.
• Learn to think functionally + concurrently
  – Tail recursion, functional datastructures, higher-order functions.
  – New processes really are that cheap.
• Simple options can go a long, long way
  – Kernelpoll
  – Bind schedulers to cores
Lessons: OTP

- Use OTP religiously
  - Use gen_servers / supervisors
  - Avoid touching `receive` / `!`
  - Avoid touching `spawn/spawn_link, trap_exit`
  - Split reused components into their own OTP applications

- Try to keep modules small, and either
  - Non side-effecting / sequential
  - An OTP behaviour (gen_server, supervisor etc.)
Lessons: KIS(S)

• Avoid..
  – Inter-node dependencies
    • Even though Erlang makes it easy..
    • Avoid having nodes with special responsibilities
    • Expect high latency / inter-node network issues
  – Complex inter-process dependencies
    • Be very afraid of processes which all rely on each other
    • Casts instead of calls.
Lessons: Bottleneck processes

• If a process receives many messages
  – Create a pool of them
  – Make sure they don’t do much intensive work
  – Manually purge message queue?

• If a process does actual work
  – Make sure it’s left alone to do it
  – and it decides when it wants to do more

• Example
  – Logging, metrics.
Lessons: use ETS

• Standard solution to many in-memory storage problems
  – Blisteringly fast
  – Linked to process (automatic cleanup)
  – No monster crashdumps
  – Avoids single-process bottlenecks

• Know its limitations..
  – Try not to reinvent mnesia
  – Distributed copies of ETS tables? Explicit indexes?
Lessons: Use Mnesia... with care

• Extremely powerful
  – Distributed, fragmentation, atomicity, transactional
  – One of the main reasons we moved to Erlang

• But *complex*
  – A lot of subtle, custom code written for error cases
    • Partitioned network; node death; fragment distribution

• mnesia ~= traditional RDBMS?
  – Powerful, fully featured… but so complex, you’ll swear and pull your hair out at times.
  – ETS: Simple, fast… but will at times lack the tools you need.
Lessons: Testing/Profiling

• Automated tests
  – Have them, and try to respect them
  – We use eunit
  – Make it easy to test a full cluster
  – Rolled our own system for stubbing out modules

• Kill random erlang processes
  – because something else almost certainly will

• Pay attention to the dialyzer and fprof

• Nothing beats heavy-duty end-to-end loadtests
  – Simulate 2 million users!
Lessons: Miscellaneous

- Obvious, but .. keep your clusters apart
  - Different VLANs, cookies
- Beware sharing cores with other OS processes
- Process priorities
  - 10,000 relatively unimportant processes running *slightly* inefficiently will clobber one vital process
- Hot swaps and code replacement:
  - Amazing, but often more effort than it’s worth
- In case things go wrong..
  - Add kill-switches, metrics and graphs for everything
  - Have a collection of helper tools, scripts.
  - Get used to using remote shells
Lessons: Be polite

• Your co-workers don’t all care about Erlang like you do
  − Just three/four Erlang developers in Demonware
• Don’t force the user of your software to
  − Use Erlang syntax
  − Read Erlang crashdumps
  − Have to understand erlang code
• Either
  − Make them all converts
  − Accept that it’s a niche language in the company
Some things we’d love to see in Erlang
Mnesia improvements?

• An Mnesia that lives and breathes network outages and node crashes.
  – Mnesia-Cassandra hybrid?
  – Eventual consistency
  – Automatic rebalancing
  – CAP theorem says there’s no magic bullet.

• Automatic clean up logic
  – Mnesia data divorced from process responsible for it
  – linking of rows to processes/nodes?
  – Distinguishing old and new incarnations of a node.
A neater OTP interface?

- `receive, !, link, spawn` is the Erlang “assembly language”
  - But you have still have to know how it works.

- More flexible supervision trees
  - Hand-crafted dependencies
    - Instead of complex nesting of `one_for_one`, `rest_for_one`, etc.
  - Hand-crafted restart strategies
    - Exponential backoffs?
  - Wrap process monitoring too?

- Processes should respond to system messages quickly
  - Writing well-behaved blocking / busy processes is messy
  - `gen_background_script`?
Easier inter-language integration?

- Erlang isn’t a general purpose language
  - It’s great for any hard, concurrency problem
  - But we would never use it for business logic
  - The ease of concurrency doesn’t make up for the difficulty in interfacing with other languages.
  - It’s too easy to just muddle through without Erlang

- Make it easy for scripts to be an erlang process
  - Standardise a subset of the protocol.
  - jinterface, twotp, rinterface etc.
Static Types, Dynamic Hacks?

• A statically typed sub-language
  – A more expressive, less forgiving Dialyzer
  – No side-effecting allowed
    • Confined to modules, helper code that is sequential
  – Being able to enable run-time warnings for dialyzer errors?

• More dynamic features
  – Possible to monkeypatch functions?
  – Easier viewing/modification of running processes.
  – Grotesque hacks are sometimes needed.
A Gentler Learning Curve?

• In Erlang
  – (Very) hard things are possible..
  – But (very) easy things still aren’t easy
  – Moving to Erlang is a big commitment
  – Have to first get through the sequential language.

• So, all the usuals
  – Standard guides, coding styles
  – Documentation aimed at non-experts
  – Friendly syntax

• A simple single-step, clustered OTP server?
  – .. easy to understand, and written the right way.
What we love about Erlang
Pretty much everything else..

• But in particular..
  – Effortless concurrency
    • The complete solution for hard concurrent problems.
  – Open source
    • We can look under the hood and play around
  – Remote shells
    • An absolute life-saver.
  – Its sheer robustness and reliability
    • Many months of uptime is par for the course
Black Ops – 24 hour stats
In short

• Erlang helps make 10s of millions of gamers happier across the world
• In Demonware, if gamers are happy then so are we.
In short
And finally..

We’re hiring!

See [http://www.demonware.net](http://www.demonware.net) for details

Thanks for listening - any questions?