erlang at hover.in

Bhasker V Kode
co-founder & CTO at hover.in

at Erlang Factory, London
June 26, 2009

brief introduction to hover.in

choose words from your blog, & decide what content / ad you want when you hover* over it

* or other events like click,right click,etc

or...

the worlds first publisher driven in-text content & ad delivery platform...

or

lets web publishers push client-side event handling to the cloud, to run various rich applications called *hoverlets*

demo at http://start.hover.in/ and http://hover.in/demo

more at http://hover.in , http://developers.hover.in/blog/

```
<html> <body>
 <a href="http://facebook.com" title="
http://onclick.hover.in/hoverlet/hover.in/crunchbase/facebook" > facebook
profile from crunchbase </a>

 <a title="http://onclick.hover.in/hoverlet/hover.in/twittersearch/election"
href=”#”>election tweets</a>

 <a href="#" title="http://onclick.hover.in/hoverlet/hover.in/nytimes/iphone"
> iphone on NYT</a>

 <a title="http://onclick.hover.in/hoverlet/hover.in/relatedyoutube/cooking"
href=”#” >cooking videos</a>  from youtube.

 <script src="http://start.hover.in/script" id="hi_start"  type="text/javascript"></script>
 </body><html>
```

- hover.in founded late 2007

- hover.in founded late 2007
- the web ~ 10- 20 years old

- hover.in founded late 2007
- the web ~ 10- 20 years old
- humans 100's of thousands of years

- hover.in founded late 2007
- the web ~ 10- 20 years old
- humans 100's of thousands of years
- but **bacteria**.... around for millions of years
  ... so this talk is going to be about what we can learn from **bacteria**, the **brain**, and **memory** in a concurrent world followed by hover.in's erlang setup and lessons learnt

```
%% Should_we_glow_underwater.erl
%% query each cell's protein and if
%% total +ves pass some value, then glow!

%% process spawned to hold state of +ves
Pid = spawn(fun()->
      should_we_glow(0,length(Cells),0)
  end)

%% querying cell protein ,message passing
[ Pid !{in, Cell } || Cell < Cells ]
```

```
should_we_glow(Ctr, Max, Acc)->
  receive
   {in,Cell}->
     One_or_zero  = should_i_glow(Cell),
     case Ctr of
       Max -> done;
       _ ->
        case Acc of
          ?SOME_VAL ->
               glow_for_8_hours(), done;
          _ ->
          NewAcc = One_or_zero + Acc,
          should_we_glow(Ctr+1,Max,NewAcc)
        end;
   _ -> error
end
```

some traits of bacteria

- each bacteria cell spawns its own proteins
- All bacteria have some sort of some presence & replies associated, *(asynchronous comm.)*
- group dynamics exhibits '*list fold'* ish operation
- only when the **Accumulator** is > some guard clause, will group-dynamics of making light (bioluminiscence) work *(eg: in deep sea)*

spawning, in practice

- for a single **google search result**, the same requests are sent to multiple machines(~1000 as of 09), which ever replies the quickest wins.
- in **amazon's dynamo architecture** that powers S3, use a (3,2,2) rule . ie Maintain 3 copies of the same data, reads/writes are succesful only when 2 concurrent requests succeed. This ratio varies based on SLA, internal vs public service. (*more on conflict resolution...*)

pattern matching behaviour

- each molecule connects to its specific receptor protein to complete the missing piece,to trigger the group behaviour that are only succesful when all of the cells participate in unison.
- Type = case UserType of

```
user -> true;
admin -> true;
_Else  -> false
end
```

supervisors, workers

- as bacteria grow, they split into two. when muscle tears, it knows exactly what to replace.
- erlang supervisors can decide restart policies: if one worker fails, restart all .... or if one worker fails, restart just that worker, more tweaks.
- can spawn multiple workers on the fly, much like the need for launching a new ec2 instant

inter-species communication

- if you look at your skin – consists of very many different species, but all bacteria found to communicate using one common chemical language.

inter-species communication

- if you look at your skin – consists of very many different species, but all bacteria found to communicate using one common chemical language.
  hmmmmmmmmmmmmmmmmm.....
  ....serialization ?!
  ....a common protein interpretor ?!
  ....or perhaps just-in-time protein compilation?!

interspecies comm. in practice

- attempts at **serialization** , cross language communication include:
  - **thrift** (by facebook)
  - **protocol buffers** (by google)
  - en/decoding , port based communication (erlang<->python at hover.in)
  - rabbitMQ shows speeds of several thousands of msgs/sec between python <-> erlang (by using...?)

talking about scaling

The brain of the worker honeybee weighs about **1mg** (~ 950,000 neurons)

- Flies acrobatically , recognizes patterns, navigates , communicates, etc
- Energy consumption: 10–15 J/op, at least 106 more efficient than digital silicon neurons

the human brain

- 100 billion neurons, stores ~100 TB
- Differential analysis e.g., we compute color
- Multiple inputs: sight, sound, taste, smell, touch
- Facial recognition subcircuits, peripheral vision
- in essence - the left & right brain vary in:
  left -> persistent disk , handles past/future
  right -> temporal caches! , handles present

in-memory is the new embedded

- servers as of '09 typically have 4 - 16 GB RAM
- stats of how companies are adding nodes

in-memory is the new embedded

- caching systems avoid disk/db makes sense
- caching systems for processing tasks makes sense
- **but ....**
- keeping your entire data in-memory by having N number of nodes ?

in-memory is the new embedded

- keeping your entire data in-memory by having N number of nodes , (where N = total data in gb / max ram per node) is like ...
  - *building a billion dollar company with 999 milion dollars of funding!*
    or
  - *having only a right brain !*
- surely we can do better than that!

in-memory capacity planning

- No matter how many machines you have, and how many cores, in production level – your product could be defined by how well you design your in-memory / RAM strategies.
- alternatives to avoid swapping could be – just leaving results partioned on diff nodes, or additional tasks to reduce the data-load further until they can fit in memory

in-memory capacity planning

- parallizing jobs in-memory is a lot of fun...
- **but...**
- more often bottleneck will not be how well you can paralliize, but how much you need to parallize so that memory does'nt swap (eg: || db reads)

#1 shard thy data to make it sufficiently un-related

- typical web backends – all user data in one table – then clustering just splits that on artibary basis. *eg: query user table where id=user1,*
- what if you have N concurrent process's accessing N diff user tables – no locks, you can ||'ze & results can come back asynchronously since sufficiently un-related.
- Warning: but more atoms (list_to_atom atoms aren't garbage collected)

#2 implementing flowcontrol

- great to handle both bursts or silent traffic & to determine bottlenecks.(eg ur own,rabbitmq,etc)
- **eg1:** when we addjobs to the queue, if it takes greater than X consistently we move it to high traffic bracket, do things differently, possibly add workers **or** ignore based on the task.
- **eg2:** amazon shopping carts, are known to be extra resilient to write failures, (dont mind multiple versions of them over time)

#3 all data is important, but some less important

- priority queue used to built heat-seeking algo (priority to crawl webpages that get more hits rather than depth-first or breadth-first)
- can configure max number of buckets
- can configure max number of urls per bucket
- can configure pyramid like queue. (moving from lower buckets to higher is easier than moving from high to higher)

erlang in a crawler architecture ?

- each time a hit occurs for a url, it moves from bucket N to bucket N+1

erlang in a crawler architecture ?

- each time a hit repeats for a URL , it moves from bucket N to bucket N+1
- crawls happen from top down (priority queue)

erlang in a crawler architecture ?

- each time a hit repeats for a URL , it moves from bucket N to bucket N+1
- crawls happen from top down (priority queue)
- so the bucket is locked, so that locked urls dont keep move up anymore

erlang in a crawler architecture ?

- each time a hit repeats for a URL , it moves from bucket N to bucket N+1
- crawls happen from top down (priority queue)
- so the bucket is locked, so that locked urls dont keep move up anymore
- each user/site has their own priority queues, which keep shifting round-robin after every X urls crawled per user/site

erlang in a crawler architecture ?

- each time a hit repeats for a URL , it moves from bucket N to bucket N+1
- crawls happen from top down (priority queue)
- so the bucket is locked, so that locked urls dont keep move up anymore
- each user/site has their own priority queues, which keep shifting round-robin after every X urls crawled per user/site
- python crawler leaves text files which dirty loaded into fragmented mnesia

#3 time spent x RAM utilization = a constant

eg: of || db reads

#4 before every succesful persistent write & after every succesful persistent read is an in-memory one

eg: hi_cache_worker's used to build most recent queue's

#5 before every succesful persistent write & after every succesful persistent read is an in-memory one

- you listen to a phone number in batch's of 3 or 4 digits. the part that absorbs just before writing (temporal), until you write into your contact book or memorize it (persistent)
- eg: if LRU cache exists in-memory, like 100 most recent url's or tags, then no need to parse server logs for computation, try during writes itself . No logs, no files. live buzz analytics!

#6 know thy RAM, trial/error to find ideal dataload

- eg: || db reads if || happens so fast, mem probs
- replication **vs** location transparency, are they fragmented, are some nodes read-only ? (rpc...)
- need metadata for which node to acess for user1, (or use hashing fn like memcache)
- are tables in-memory (right brain), cached from disk , or on disk alone (left brain)
- fortunately **mnesia** allows highly granular choices

#7 what cannot be measured cannot be improved

- you can't improve what you can't measure. an investment in debugging utilities is a good investment
- looking forward to debugging with dtrace,gproc etc but until then – just a set/get away!
- using **tsung** (written in erlang again) – load performance testing tool, for simulating 100's of concurrent users/requests , and great for analysing bottlenecks of your system ,CDN's) ,

hi_cache_worker

- a circular queue implemented via gen_server
- set (ID , Key , Value , OptionsList)

  Options are {purge, <true| false>}

  { size , <integer> }

  { set_callback , <Function> }

  { delete_callback , <Function> }

  { get_callback , <Function> }

  { timeout, <int>, <Function> }

  ID is usually a siteid or “global”

- C = hi_cache_worker,
  C:set (User1, “recent_saved” , Value)
  C:set (“global”, “recent_hits” , Value
  [{size,1000}])

  C:get (“global”,”recent_voted”)
  C:get (User1,”recenthits”)
  C:get (User1,”recent_cron_times”)
- (Note: initially used in debugging internally -> then reporting -> next in public community stats)

7 rules of in-memory capacity planning

(1) shard thy data to make it sufficiently un-related

(2) implementing flowcontrol

(3) all data is important, but some less important

(4) time spent x RAM utilization = a constant

(5) before every succesful persistent write & after every succesful persistent read is an in-memory one

(6) know thy RAM, trial/error to find ideal dataload

(7) what cannot be measured cannot be improved

summary of erlang at hover.in

- LYME stack since ~dec 07 , 3 nodes (64-bit 4gb)
- python crawler, associated NLP parsers, cpu time-splicing algo's for cron's app, configurable priority queue's for heat-seeking algo's app, flowcontrol app , caching app , pagination app for memoizing
- remote node debugger, cyclic queue workers, lru cache workers , headless-firefox for thumbnails
- touched 1 million hovers/month in May'09 after launching closed beta to publishers in Jan 09

summary of our erlang modules

rewrites.erl error.erl frag_mnesia.erl hi_api_response.erl hi_appmods_api_user.erl hi_cache_app.erl , hi_cache_sup.erl hoverlingo.erl hi_cache_worker.erl hi_lru_worker.erl hi_classes.erl hi_community.erl hi_cron_hoverletupdater_app.erl hi_cron_hoverletupdater.erl hi_cron_hoverletupdater_sup.erl hi_cron_kwebucket.erl hi_crypto.erl hi_flowcontrol_hoverletupdater.erl hi_htmlutils_site.erl hi_hybridq_app.erl hi_hybridq_sup.erl hi_hybridq_worker.erl hi_login.erl hi_mailer.erl hi_messaging_app.erl hi_messaging_sup.erl hi_messaging_worker.erl hi_mgr_crawler.erl hi_mgr_db_console.erl hi_mgr_db.erl hi_mgr_db_mnesia.erl hi_mgr_hoverlet.erl hi_mgr_kw.erl hi_mgr_node.erl hi_mgr_thumbs.erl hi_mgr_traffic.erl hi_nlp.erl hi_normalizer.erl hi_pagination_app.erl hi_pagination_sup.erl, hi_pagination_worker.erl hi_pmap.erl hi_register_app.erl hi_register.erl, hi_register_sup.erl, hi_register_worker.erl hi_render_hoverlet_worker.erl hi_rrd.erl , hi_rrd_worker.erl hi_settings.erl hi_sid.erl hi_site.erl hi_stat.erl hi_stats_distribution.erl hi_stats_overview.erl hi_str.erl hi_trees.erl hi_utf8.erl hi_yaws.erl

references

- http://developers.hover.in
- http://erlang.org
- http://memcached.org , http://rabbitmq.org/ http://highscalability.com/
- http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pd
- shoutout to everyone at #erlang !
- amazing brain-related talks at http://ted.com ,
- go read more about the brain and hack on erlang NOW!





thank you

http://developers.hover.in