Load Regulation in Erlang

A Practical Application

Dave Smith
Basho Technologies

First, a story...

- Two Engineers...
- A bit of Erlang tracing code...
- Writing to a zlib'd file...

Why Regulation?

- Load can shift rapidly
  - Bursty traffic patterns
  - Hardware/software failure
- Layer of defense

Why Riak?

- Distributed Erlang
- Reputation for performance and predictability
- Want to push envelope on failing gracefully

JOBs

- Queue
  - Time/Size Restriction
- Regulator
  - Counter
  - Rate-based

Applied JOBs

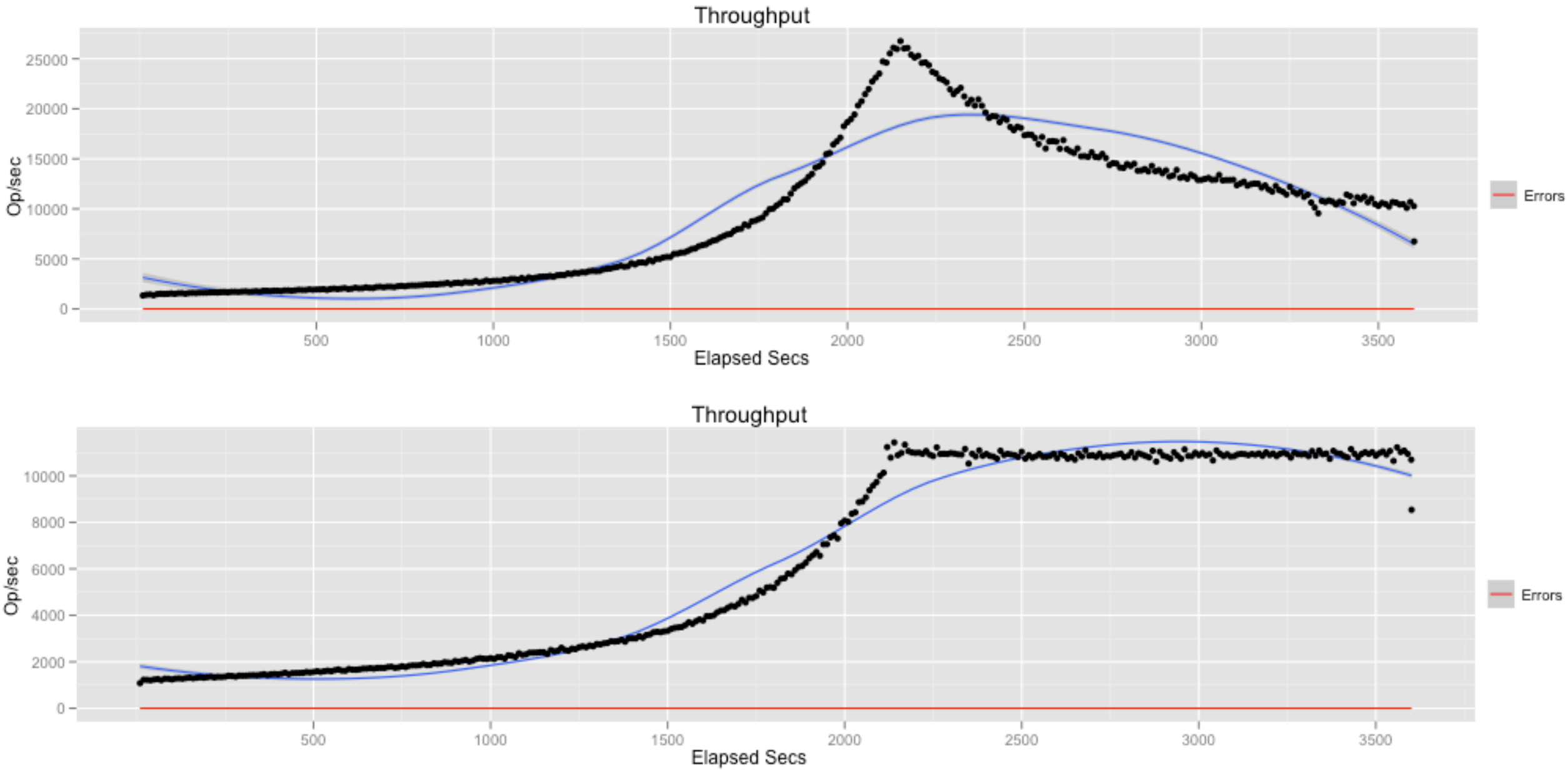
- Simple integration w/ existing APIs
- Restricted to GET/PUT operations
- Evaluated max-rate and max-concurrent

Code (simple!)

```
case jobs:ask(riak_kv_fsm) of
  {ok, JobId} ->
    try
      {ok, Pid} = riak_kv_get_fsm_sup:start_get_fsm(...),
      Timeout = recv_timeout(Options),
      wait_for_reqid(ReqId, Timeout)
    after
      jobs:done(JobId)
    end;

  {error, rejected} ->                  %% Overload!
    {error, timeout}
end
```

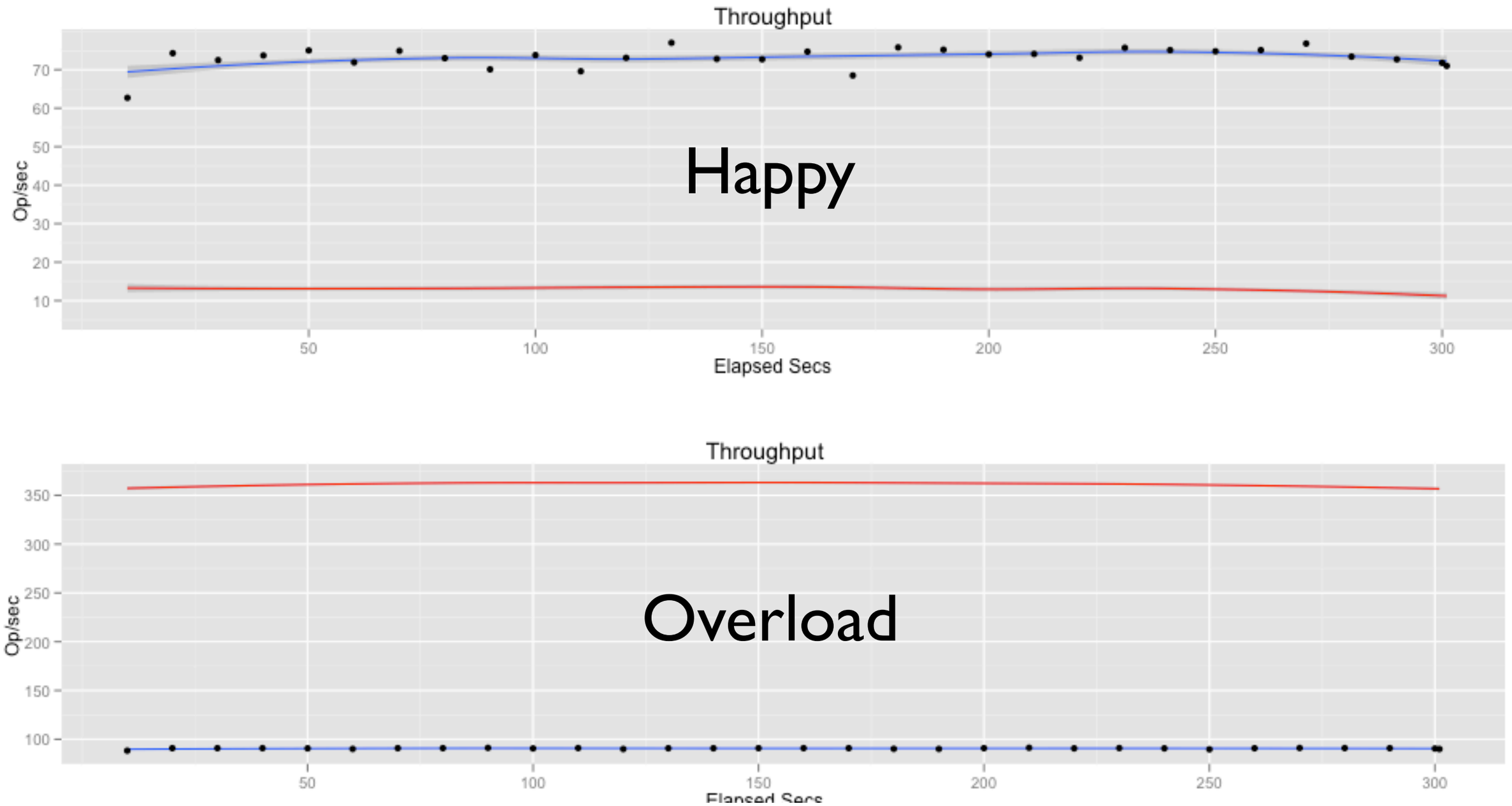
Approve vs. Rate



Observations

- Low overhead (3-4 ms under duress)
- Improves latency profile (flatter)
- May improve sustainable throughput

Overload Throughput



Overload Latency