Options for Predictive Server Analytics

Joe Conway and Jeff Hamann

credativ Group and Forest Informatics

12/12/12

Server Monitoring with Predictive Analytics

- Results of investigation
  - PostgreSQL, R, and PL/R
  - Server monitoring application
  - Performing predictive analytics of performance
- Usually server monitoring is reactive
  - Threshold is exceeded
  - Alert sent
  - Something bad has already happened
- Investigate feasibility of proactive server management
  - Dynamic statistical analysis

Intro to PL/R

What is PL/R?

- R Procedural Language for PostgreSQL. Enables user-defined SQL functions to be written in the R language

What is R?

- R is an open source (GPL) language and environment for statistical computing and graphics. R provides a wide variety of statistical (linear and nonlinear modeling, classical statistical tests, time-series analysis, classification, clustering, ...) and graphical techniques, and is highly extensible.

http://www.r-project.org

http://www.joeconway.com/plr

Pros

- Leverage people's knowledge and skills
  - statistics/math is a specialty
- Leverage hardware
  - server better able to handle analysis of large datasets
- Processing/bandwidth efficiency
  - why send large datasets across the network?
- Consistency of analysis
  - ensure analysis done consistently once vetted
- Abstraction of complexity
  - keep system understandable and maintainable
- Leverage R
  - rich core functionality and huge ecosystem

Data Collection, Processing, and Storage

- Keep it simple: use cron
- First test: PL/pgSQL and native PostgreSQL arrays
- Second test: PL/R and R objects
- Both cases: use PL/R to process collected data

What to Collect?

- Active and Total sessions
- Blocks fetched and Blocks hit
- Cache hit fraction
- Lock waits
- Mem free and Mem cached
- Swap free
- IO wait and Cpu idle
- Blocks read and written per second
- Blocks read and written
- Capture time

Cache Hit Fraction

```
CREATE OR REPLACE FUNCTION cache_hit_fraction()
RETURNS float8 AS $$
  WITH
  db AS
  (SELECT oid FROM pg_database WHERE datname = current_database()),
  bh AS
  (SELECT pg_stat_get_db_blocks_hit((SELECT oid FROM db))::float8 as bh),
  bf AS
  (SELECT pg_stat_get_db_blocks_fetched((SELECT oid FROM db))::float8 as bf)
  SELECT
    CASE WHEN (SELECT bf FROM bf) > 0 THEN
      ((SELECT bh FROM bh) / (SELECT bf FROM bf))::float8
    ELSE
      0.0
    END AS cache_hit_fraction
$$ LANGUAGE sql;
```

Memory Info

```
CREATE EXTENSION plr;

CREATE OR REPLACE FUNCTION r_meminfo()
RETURNS SETOF text AS $$
  system("cat /proc/meminfo",intern=TRUE)
$$ LANGUAGE plr;

CREATE OR REPLACE FUNCTION meminfo(OUT metric text, OUT val bigint)
RETURNS SETOF record AS $$
  select trim(split_part(r_meminfo(),':',1))
         as metric,
         split_part(trim(split_part(r_meminfo(),':',2)),' ',1)::bigint
         as val;
$$ LANGUAGE sql;

SELECT * FROM meminfo();
```

IO Wait and CPU Idle

```
CREATE OR REPLACE FUNCTION r_iostat_c() RETURNS text AS $$
  res<-system("iostat -c",intern=TRUE)
  finres<-gsub(" +", " ", res[4])
  return(finres)
$$ LANGUAGE plr;

CREATE OR REPLACE FUNCTION iowait() RETURNS float8 AS $$
  select split_part(trim(r_iostat_c()),' ',4)::float8 as iowait;
$$ LANGUAGE sql;

CREATE OR REPLACE FUNCTION cpu_idle() RETURNS float8 AS $$
  select split_part(trim(r_iostat_c()),' ',6)::float8 as cpu_idle;
$$ LANGUAGE sql;
```

Disk IO

```
CREATE OR REPLACE FUNCTION r_iostat_d(
  OUT device text, OUT tps float8, OUT blk_read_p_s float8,
  OUT blk_wrtn_p_s float8, OUT blk_read bigint ,OUT blk_wrtn bigint
) RETURNS SETOF record AS $$
  res<-system("iostat -d",intern=TRUE)
  res<-res[4:(length(res)-1)]
  finres<-gsub(" +", " ", res)
  ffinres<-vector(mode="character")
  for (i in 1:length(finres)) {
    ffinres <- rbind(ffinres, unlist(strsplit(finres[i], " ")))}
  if (length(finres) == 1) {
    fdf <-data.frame(ffinres[1], as.numeric(ffinres[2]),
                     as.numeric(ffinres[3]), as.numeric(ffinres[4]),
                     as.numeric(ffinres[5]), as.numeric(ffinres[6]))}
  else {
    fdf <-data.frame(ffinres[,1], as.numeric(ffinres[,2]),
                     as.numeric(ffinres[,3]), as.numeric(ffinres[,4]),
                     as.numeric(ffinres[,5]), as.numeric(ffinres[,6]))}
  return(fdf) $$ LANGUAGE plr;
```

Blocks per Second

```
CREATE OR REPLACE FUNCTION blk_read_p_s(device text)
RETURNS float8 AS $$
  select blk_read_p_s FROM r_iostat_d() where device = $1;
$$ LANGUAGE sql;

CREATE OR REPLACE FUNCTION blk_wrtn_p_s(device text)
RETURNS float8 AS $$
  select blk_wrtn_p_s FROM r_iostat_d() where device = $1;
$$ LANGUAGE sql;
```

Blocks Read/Written

```
CREATE OR REPLACE FUNCTION blk_read(device text)
RETURNS bigint AS $$
  select blk_read FROM r_iostat_d() where device = $1;
$$ LANGUAGE sql;

CREATE OR REPLACE FUNCTION blk_wrtn(device text)
RETURNS bigint AS $$
  select blk_wrtn FROM r_iostat_d() where device = $1;
$$ LANGUAGE sql;
```

Putting it All Together

```
-- assume metric id is index into
-- single dimensional observation array
CREATE TABLE metrics (
  id int primary key,
  cum bool default false,
  metric text not null,
  sql text not null
);
CREATE UNIQUE INDEX metrics_uidx ON metrics(metric);
```

Putting it All Together (cont.)

```
INSERT INTO metrics VALUES
 (1,DEFAULT,'active sessions',
  'select count(1) from pg_stat_activity
   where state != $$idle$$ and pid != pg_backend_pid()')
,(2,DEFAULT,'total sessions',
  'select count(1) from pg_stat_activity')
,(3,DEFAULT,'blocks fetched',
  'select pg_stat_get_db_blocks_fetched(
   (select oid from pg_database where datname = current_database()))')
,(4,DEFAULT,'blocks hit',
  'select pg_stat_get_db_blocks_hit(
   (select oid from pg_database where datname = current_database()))');
```

Putting it All Together (cont.)

```
INSERT INTO metrics VALUES
 (5,DEFAULT,
  'cache hit fraction','select cache_hit_fraction()')
,(6,DEFAULT,'lock waits',
  'select count(1) from pg_locks where not granted')
,(7,DEFAULT,'mem free',
  'select val from meminfo() where metric = $$MemFree$$')
,(8,DEFAULT,'mem cached',
  'select val from meminfo() where metric = $$Cached$$');
```

Putting it All Together (cont.)

```
INSERT INTO metrics VALUES
 (9,DEFAULT,'swap free',
  'select val from meminfo() where metric = $$SwapFree$$')
,(10,DEFAULT,'iowait',
  'select iowait()')
,(11,DEFAULT,'cpu_idle',
  'select cpu_idle()')
,(12,DEFAULT,'blk_read_p_s',
  'select blk_read_p_s($$sdb$$)');
   --adjust device name for given server
```

Putting it All Together (cont.)

```
--adjust device names for given server
INSERT INTO metrics VALUES
 (13,DEFAULT,'blk_wrtn_p_s',
  'select blk_wrtn_p_s($$sdb$$)')
,(14,DEFAULT,'blk_read',
  'select blk_read($$sdb$$)')
,(15,DEFAULT,'blk_wrtn',
  'select blk_wrtn($$sdb$$)')
,(32,DEFAULT,'capture_time',
  '');
```

Metrics Storage

```
CREATE TABLE measurement (
  ts timestamp without time zone primary key,
  vals float8[] not null
);
```

Metrics Collection Function

```
CREATE OR REPLACE FUNCTION capture_all_metrics() RETURNS float8 AS $$
  DECLARE
    rec          record;
    res          float8;
    vals         float8[];
    st           timestamp without time zone;
    et           timestamp without time zone;
  BEGIN
    st := clock_timestamp();
    FOR rec IN SELECT id, metric, sql FROM metrics
               WHERE id < 32 ORDER BY id LOOP
      EXECUTE rec.sql INTO res;
      vals[rec.id] := res;
    END LOOP;
    et := clock_timestamp();
    vals[32] := extract(seconds from (et - st))::float8;
    INSERT INTO measurement VALUES (st, vals);
    PERFORM pg_stat_reset();
    RETURN vals[32];
  END; $$ LANGUAGE plpgsql;
```

Metrics Collection cron

```
* * * * * su - postgres -c \
    "psql pgbench -c 'SELECT capture_all_metrics()'" \
    > /dev/null
```

Metrics Storage

```
CREATE TABLE measurement_robj (
  ts timestamp without time zone primary key,
  samplegrp bytea not null
);
```

Metrics Collection Function

```
CREATE OR REPLACE FUNCTION capture_all_metrics(grpsize int, deltasecs int)
RETURNS bytea AS $$
  ## Next line only used in interactive R session
  # require(RPostgreSQL)

  ## Initialize vals matrix and tms vector
  vals <- matrix(nrow = grpsize, ncol=32)
  tms <- array(dim = grpsize)
```

Metrics Collection Function (cont.)

```
## Connect to Postgres database
## Actually a noop in PL/R
drv <- dbDriver("PostgreSQL")
conn <- dbConnect(drv, user="postgres", dbname="pgbench",
                  host="localhost", port="55594")

## determine which metrics to collect
sql.str <-
  "SELECT id, metric, sql FROM metrics WHERE id < 32 ORDER BY id"
rec <- dbGetQuery(conn, sql.str)
```

Metrics Collection Function (cont.)

```
## outer loop: perform this grpsize times
for (grpi in 1:grpsize) {
  ## start out with a stats reset to attempt
  ## to get consistent sampling interval
  sql.str <- "SELECT 1 FROM pg_stat_reset()"
  retval <- dbGetQuery(conn, sql.str)

  ## sleep for sampling interval
  Sys.sleep(deltasecs)
```

Metrics Collection Function (cont.)

```
## set this measurement start time
st <- Sys.time()

## collect metric for this sample group
for (i in 1:length(rec$id)) {
  vals[grpi, rec$id[i]] <- as.numeric(dbGetQuery(conn, rec$sql[i]))
}

## set this measurement end time
et<-Sys.time()
```

Metrics Collection Function (cont.)

```
  ## calc time required for this sample
  vals[grpi, 32] <- difftime(et, st)

  ## save sample times
  tms[grpi] <- st

}  ## End of outer loop

## Initialize sample group variable
samplegrp <- NULL
samplegrp$grpsize <- grpsize
samplegrp$tms <- tms
samplegrp$vals <- vals
```

Metrics Collection Function (cont.)

```
  ## calculate sample group statistics
  ## first averages
  samplegrp$avgs <- apply(vals, 2, mean)
  ## second ranges
  samplegrp$rngs <- apply(vals, 2, max) - apply(vals, 2, min)

  ## Not required and noop in PL/R, but to be consistent with R session
  dbDisconnect(conn)
  dbUnloadDriver(drv)

  ## return the samplegrp R object
  return(samplegrp)
$$ LANGUAGE plr;
```

Metrics Collection cron

```
*/3 * * * * su - postgres -c "psql pgbench -c \
   'INSERT INTO measurement_robj \
    VALUES (current_timestamp, capture_all_metrics(3, 30))'" > /dev/null
```

Misc Utility Functions

```
CREATE OR REPLACE FUNCTION samplegrp_delta_ts(samplegrp bytea)
RETURNS float8[] AS $$
  return(samplegrp$vals[,32])
$$ LANGUAGE plr;

CREATE OR REPLACE FUNCTION samplegrp_avgs(samplegrp bytea)
RETURNS float8[] AS $$
  return(samplegrp$avgs)
$$ LANGUAGE plr;

CREATE OR REPLACE FUNCTION samplegrp_rngs(samplegrp bytea)
RETURNS float8[] AS $$
  return(samplegrp$rngs)
$$ LANGUAGE plr;
```

Simulating Steady-State Load

```
createdb pgbench
pgbench -i -s 1000 pgbench
# pgbench -c <num concurrent clients> -T <num seconds to run> pgbench
# early test runs
pgbench -c 4 -T 86400 pgbench
pgbench -c 8 -T 86400 pgbench
pgbench -c 16 -T 86400 pgbench
pgbench -c 12 -T 86400 pgbench
# latest test runs
pgbench -c 100 -T 604800 -f pgbench-custom.sql pgbench
watch -n 1 watchactiveconns.sh pgbench
```

pgbench-custom.sql

```
\set nbranches :scale
\set ntellers 10 * :scale
\set naccounts 100000 * :scale
\setrandom aid 1 :naccounts
\setrandom bid 1 :nbranches
\setrandom tid 1 :ntellers
\setrandom delta -5000 5000
BEGIN;
UPDATE pgbench_accounts SET abalance = abalance + :delta WHERE aid = :aid;
SELECT abalance FROM pgbench_accounts WHERE aid = :aid;
UPDATE pgbench_tellers SET tbalance = tbalance + :delta WHERE tid = :tid;
UPDATE pgbench_branches SET bbalance = bbalance + :delta WHERE bid = :bid;
INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VALUES (:tid, :bid, :
END;
\setrandom sleepdur 100000 200000
\sleep :sleepdur us
```

watchactiveconns.sh

```
#!/bin/bash
#
# Note: this syntax works with PostgreSQL 9.2
# Usage: watchactiveconns.sh <dbname>
# Useful with watch, e.g.
# watch -n 1 watchactiveconns.sh pgbench

psql -c "
  select count(1) from pg_stat_activity
  where state != 'idle'
  and pid != pg_backend_pid()
  and datname = '$1'
" $1
```

Simulating Transient Events

```
42 */3 * * * su - postgres -c \
  "psql pgbench -c 'select * from generate_series(1,300000000)'" \
  > /dev/null
```

Gather and Transform - Initialize

```
CREATE OR REPLACE FUNCTION samplegrp_init_qccvals()
RETURNS int AS $$
  qccvals<<-data.frame()
  return(nrow(qccvals))
$$ LANGUAGE plr;
```

Gather and Transform - Construct

```
CREATE OR REPLACE FUNCTION samplegrp_construct_qccvals
(
 samplegrp bytea
,sampletrial int
)
RETURNS int AS $$
  n <- (nrow(qccvals) / samplegrp$grpsize) + 1
  if (n <= sampletrial) {
    qccvals <<- rbind(qccvals, data.frame(samplegrp$tms,
                                          samplegrp$vals,
                                          sample=n,
                                          trial=TRUE))
  } else {
    qccvals <<- rbind(qccvals, data.frame(samplegrp$tms,
                                          samplegrp$vals,
                                          sample=n,
                                          trial=FALSE))}
  return(n)
$$ LANGUAGE plr;
```

Gather and Transform - Execute

```
SELECT samplegrp_init_qccvals();
SELECT samplegrp_construct_qccvals(touter.samplegrp, 30) FROM
  (
    SELECT tinner.ts, tinner.samplegrp FROM
    (
      SELECT ts, samplegrp FROM measurement_robj ORDER by ts DESC LIMIT 40
    ) tinner ORDER BY tinner.ts
  ) touter;
```

Gather and Transform - Results

```
CREATE OR REPLACE FUNCTION qccvals() RETURNS SETOF RECORD AS $$
  return(qccvals)
$$ LANGUAGE plr;

SELECT * FROM qccvals() AS qcc(
 tms float8,
  X1 float8,  X2 float8,  X3 float8,  X4 float8,
  X5 float8,  X6 float8,  X7 float8,  X8 float8,
  X9 float8, X10 float8, X11 float8, X12 float8,
 X13 float8, X14 float8, X15 float8, X16 float8,
 X17 float8, X18 float8, X19 float8, X20 float8,
 X21 float8, X22 float8, X23 float8, X24 float8,
 X25 float8, X26 float8, X27 float8, X28 float8,
 X29 float8, X30 float8, X31 float8, X32 float8,
 sample int,
 trial bool
);
```

The Problem

What happened?

1. It's Friday afternoon...
2. We get an email...
3. We *don't* get another weekend...

What we are looking for:

1. We monitoring several metrics (hits, mem, i/o)
2. We needed to detect changes quickly (within minutes or seconds)
3. We may need to detect for patterns (pattern recognition?)
4. We need to use PostgreSQL server friendly tools

Our Criteria

1. We want to avoid learning cliffs.
2. We want to use existing tools.
3. We want to use a repeatable process.
4. We want it to be automated.

Predictive Analytics - What does it include?

Predictive analytics includes a variety of statistical techniques including:

- Statistical modeling
- Machine learning
- Pattern matching, and
- Data mining

to analyze current and historical facts and make predictions about future events.

Predictive Analytics - Where is it used?

Predictive analytics is used in:

- actuarial science,
- marketing,
- financial services,
- insurance,
- telecommunications,
- retail,
- travel,
- healthcare,
- pharmaceuticals

Our Possible Solutions

What we are looking for:

1. We are looking for causal factors
2. We are looking for correlations
3. We are looking for leading indicators of system congestion or failures.

What we can use:

1. We can use PostgreSQL (inside server) - Joe
2. We can use R (outside server) - Jeff
3. We can use PL/R which give us the best of both worlds.

Predictive Analytics - Using CRAN Views

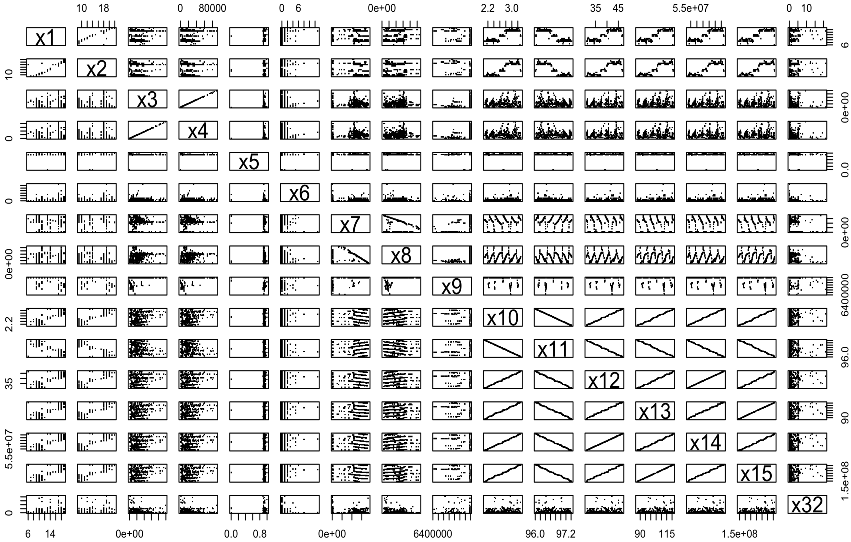
There are plenty of possible tools:

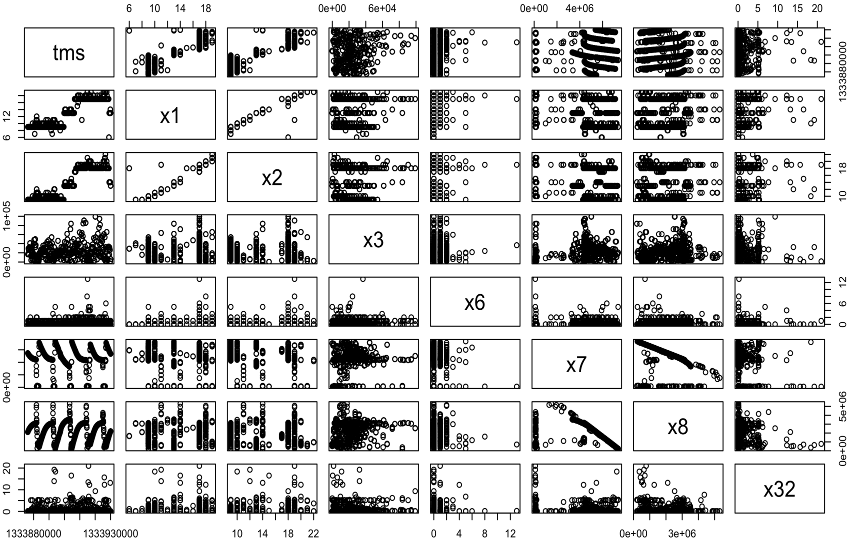
- Machine Learning
- Time Series
- Experimental Design
- Multivariate Statistics

There are plenty of possible packages in each of the CRAN Views. Check them out!

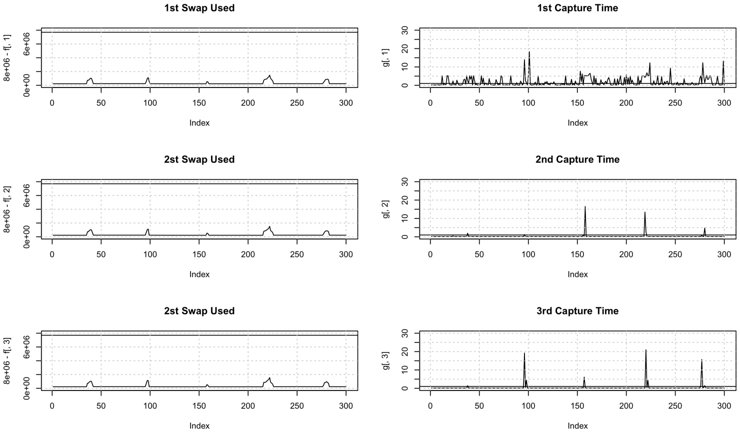
An Example Matrix Plot

```
> data <- read.data(..blah, blah, blah...)
> plot( data )
```

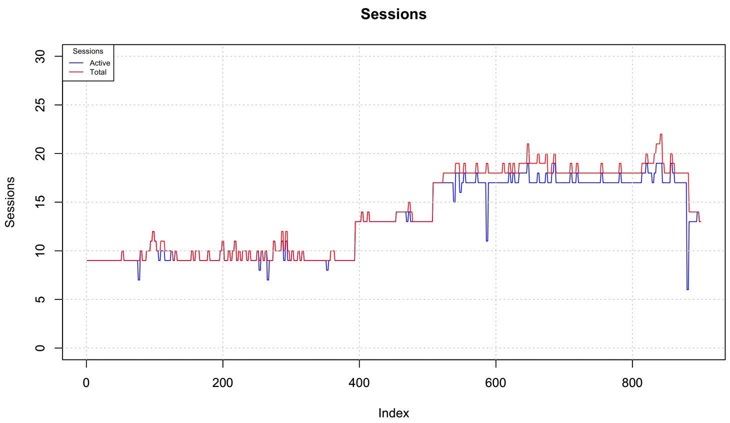

x1
x2
x3
x4
x5
x6
x7
x8
x9
x10
x11
x12
x13
x14
x15
x32
10 18
0 80000
0 6
0e+00
2.2 3.0
35 45
5.5e+07
0 10
6 14
0e+00
0.0 0.8
0e+00
6400000
96.0 97.2
90 115
1.5e+08
10
0
0
0e+00
2.2
35
5.5e+07
0
6
0e+00
0.0
0e+00
6400000
96.0
90
1.5e+08


tms
x1
x2
x3
x6
x7
x8
x32
6 10 14 18
0e+00 6e+04
0e+00 4e+06
0 5 10 15 20
1333880000
12
6
18
10
1e+05
0e+00
12
6
0
0e+00
5e+06
0e+00
20
10
0
1333880000 1333930000
10 14 18 22
0 4 8 12
0e+00 3e+06

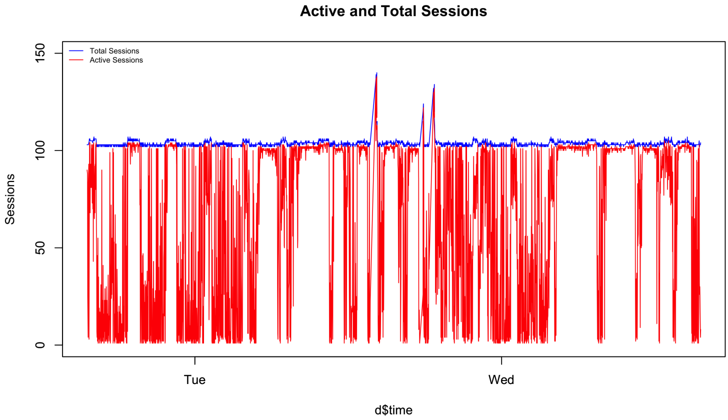
Two basic metrics



Two more metrics



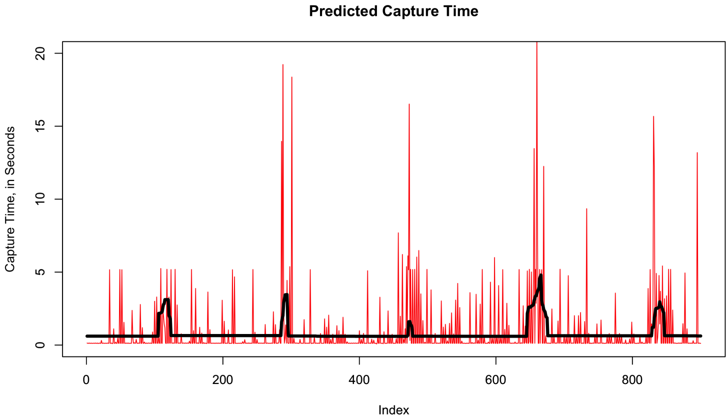
Two more metrics



```
## fit using a simple linear regression
fit <- lm( mean.cap.time ~ mean.swap.used )

## just print the summary
print( summary( fit ) )
```

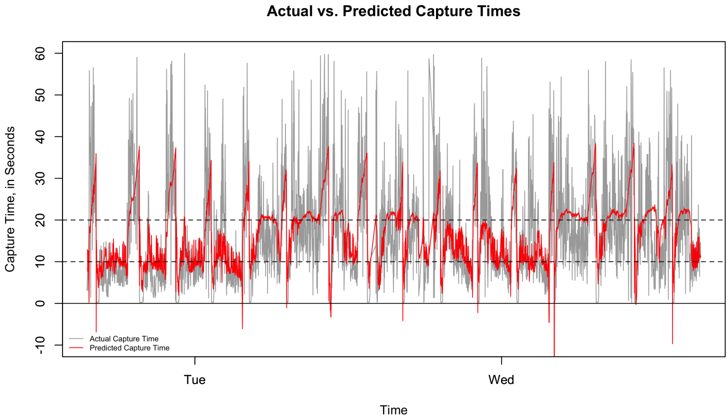
```
> summary( fit )

Call:
lm(formula = mean.cap.time ~ mean.swap.used)

Residuals:
    Min      1Q  Median      3Q     Max 
-2.6447 -0.4947 -0.3798  0.1226  9.6532 

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)    
(Intercept)     2.516e+01  2.657e+00   9.471   <2e-16 ***
mean.swap.used -3.157e-06  3.443e-07  -9.171   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 

Residual standard error: 1.146 on 298 degrees of freedom
Multiple R-squared: 0.2201, Adjusted R-squared: 0.2175 
F-statistic:  84.1 on 1 and 298 DF,  p-value: < 2.2e-16 

>
```

Capture Time Predictions



```
> summary( slm3 )

Call:
lm(formula = capture_time ~ `mem cached` + `swap free` + session.pct,
    data = d3)

Residuals:
    Min      1Q  Median      3Q     Max
-36.137  -5.824  -1.954   3.766  52.783

Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)   3.547e+01  1.152e+00   30.78   <2e-16 ***
`mem cached` -4.271e-05  2.589e-06  -16.50   <2e-16 ***
`swap free`  -2.210e-06  1.111e-07  -19.88   <2e-16 ***
session.pct   6.706e+00  6.001e-01   11.18   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 9.945 on 2807 degrees of freedom
Multiple R-squared: 0.3568, Adjusted R-squared: 0.3561
F-statistic: 519.1 on 3 and 2807 DF,  p-value: < 2.2e-16

>
```

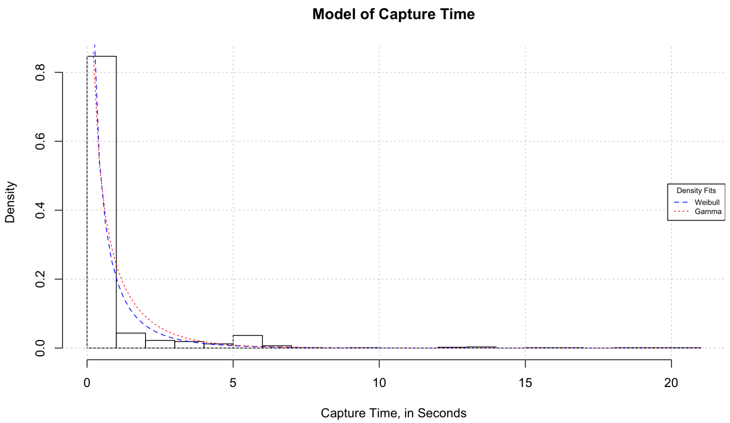
Another Model



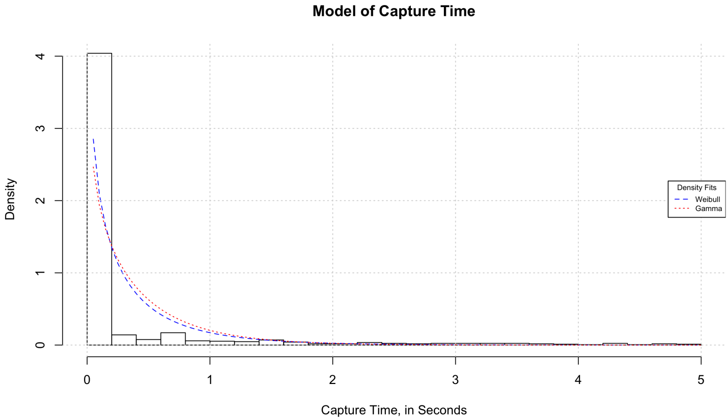
Fitting Capture Time Distributions

```
## fit a weibull distribution
d.w2 <- fitdistr( data$x32, "weibull")

## fit a gamma distribution
d.w3 <- fitdistr( data$x32, "gamma")
```

An Example of Distribution Fitting



An Example of Distribution Fitting



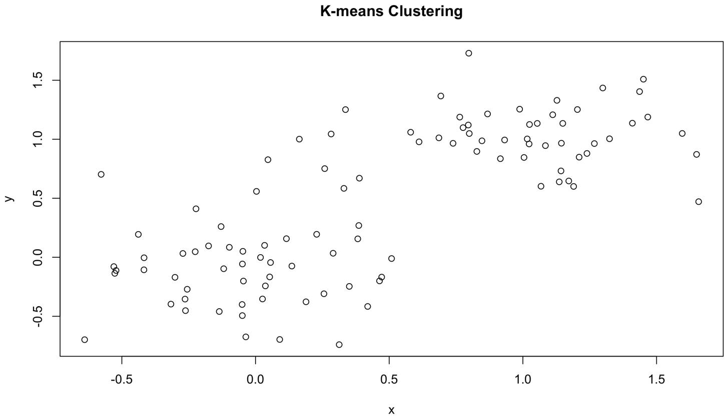
Machine Learning

1. Principal Component Analysis
2. Clustering (k-means and k-medoids)
3. Supervised learning and Unsupervised learning

An Example of K-means Clustering

```
# a 2-dimensional example
x <- rbind(matrix(rnorm(100, sd = 0.3), ncol = 2),
           matrix(rnorm(100, mean = 1, sd = 0.3),
           ncol = 2))
colnames(x) <- c("x", "y")
plot(x)
```

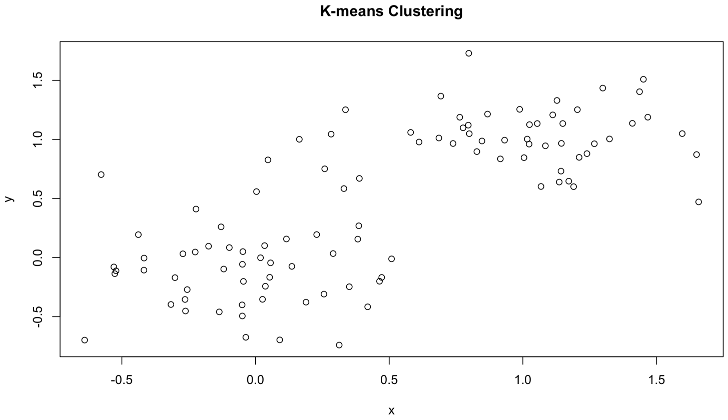
An Example of K-means Clustering



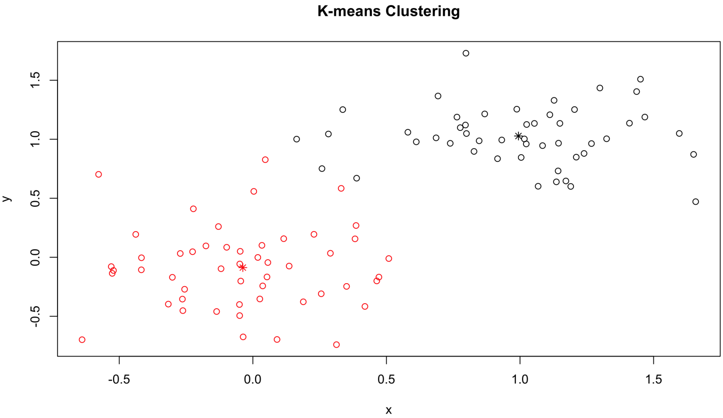
An Example of K-means Clustering

```
(cl <- kmeans(x, 2, nstart = 25))
plot(x, col = cl$cluster)
points(cl$centers, col = 1:2, pch = 8, cex=2)
```

An Example of K-means Clustering



An Example of K-means Clustering



Statistical Process Control

1. Statistical Process Control (SPC) is a time honored and well demonstrated method of process management.
2. SPC has long been used for measuring and monitoring quality in industrial manufacturing facilities.
3. SPC has been rebranded as Continuous Process Improvement (CPI) and Total Quality Management (TQM).
4. SPC is a key part of the Six Sigma and Lean Six Sigma.
5. It's simple, not easy.

The qcc Package

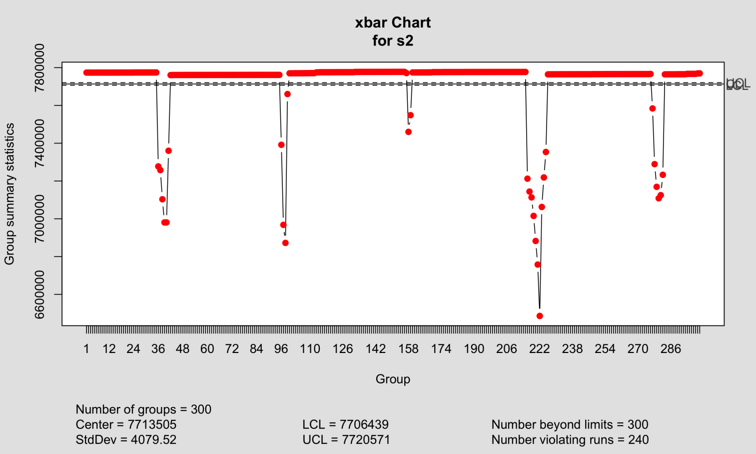
The qcc package for the R statistical environment allows users to:

- Plot Shewhart quality control charts for continuous, attribute and count data;
- Plot Cusum and EWMA charts for continuous data;
- Draw operating characteristic curves;
- Perform process capability analyses;
- Draw Pareto charts and cause-and-effect diagrams.

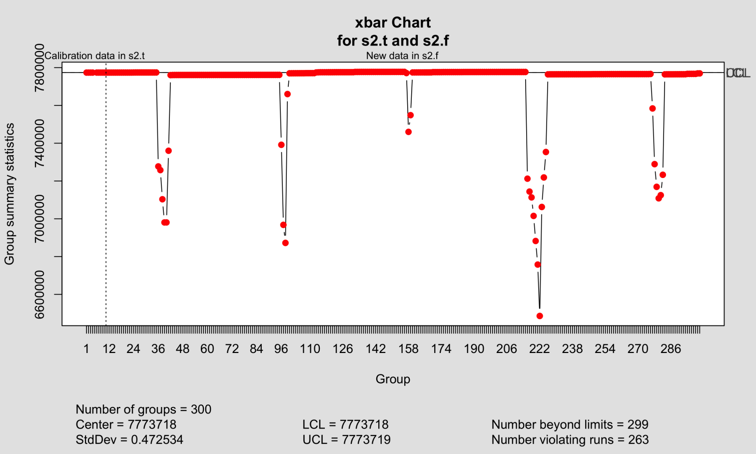
XBar Chart using qcc

```
> obj <- qcc( mean.swap.free[1:10,], type="xbar",
            newdata=mean.swap.free[11:300,] )
```

XBar Chart using qcc



XBar Chart using qcc



Our Criteria

1. Avoid learning cliffs. Maybe.
2. Use existing tools (PostgreSQL+R=PL/R).
3. Use a repeatable process (cronjobs)
4. Automated (see cronjobs).

Future Directions

- Harvest data from pg_stat_statements, pg_* tables
- Data mining for changing correlations.
- Pattern Recognition for failure prediction.
- Active control (modify postgresql.conf, firewalls)
- Polling multiple servers

Thank You

- Questions?

Joe Conway

joe.conway@credativ.com

Jeff Hamann

jeff.hamann@forestinformatics.com