Paperless™
Post

# How PostgreSQL 9 Makes Web Architecture Sweeter

Jonathan S. Katz

Vice President, Technology

http://www.paperlesspost.com

# Introduction

- No secret: PostgreSQL 9 has some very powerful new features
- How do they extend to web applications?

# Overview

- Review new features and how they relate to web apps
- Series of case studies of PostgreSQL 9 optimizations
- Overview of Sphinx vs. tsearch2 + tying into PostgreSQL 9 + web

# PostgreSQL 9: The List

- http://wiki.postgresql.org/wiki/What's_new_in_PostgreSQL_9.0

# Highlights

- JOIN removal
  - Play more nicely with ORMs
    - (ORMs + PostgreSQL – separate discussion)
- IS NOT NULL + indexes
- DEFERRABLE UNIQUE CONSTRAINTS
- Hstore improvements: “no limits”
- LISTEN / NOTIFY message passing

# Caveats

- Access to new features depends on PostgreSQL adapter
  - ActiveRecord does not support hstore
  - Nonblocking access to LISTEN / NOTIFY

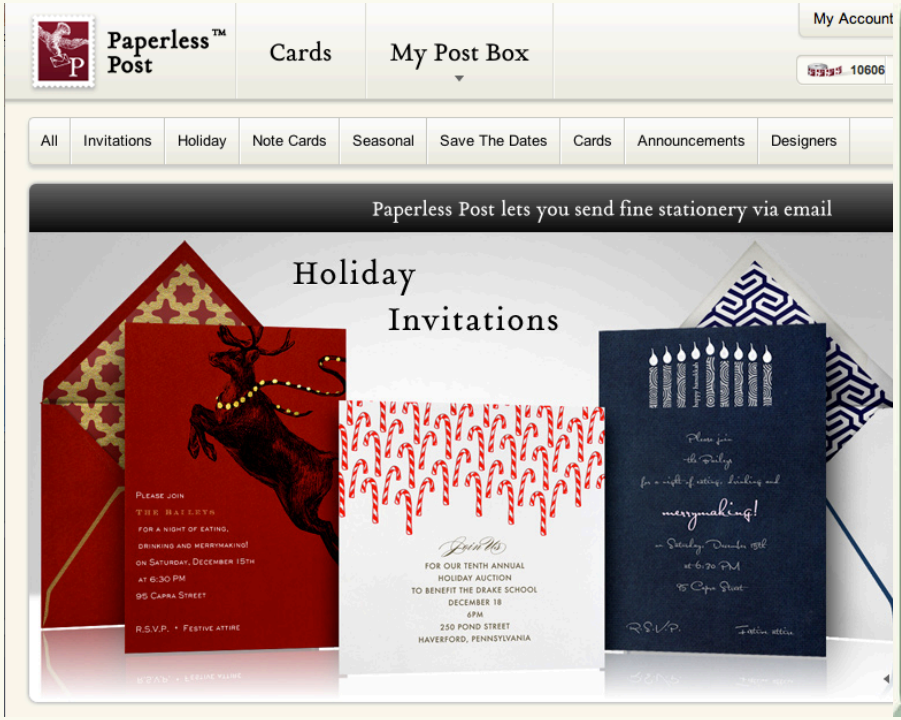
# More Highlights?

- Native Streaming Replication / Hot Standby

- Wow.

# Real World: Paperless Post

- Provides stationery designed via web interface, delivered via email




# Complex Technology Stack

- Web Servers + Load Balancing
  - nginx, haproxy, thin (Ruby app server)
- Background Workers
  - Message queues
  - Scheduled jobs
- Caching (memcache, redis)
- 

# Major Considerations

- High traffic (especially the holidays)
  - Response time
  - High availability
- Developer tools + PostgreSQL playing nicely
  - "transparent changes" in developer environment
- How does streaming replication / hot standby help?

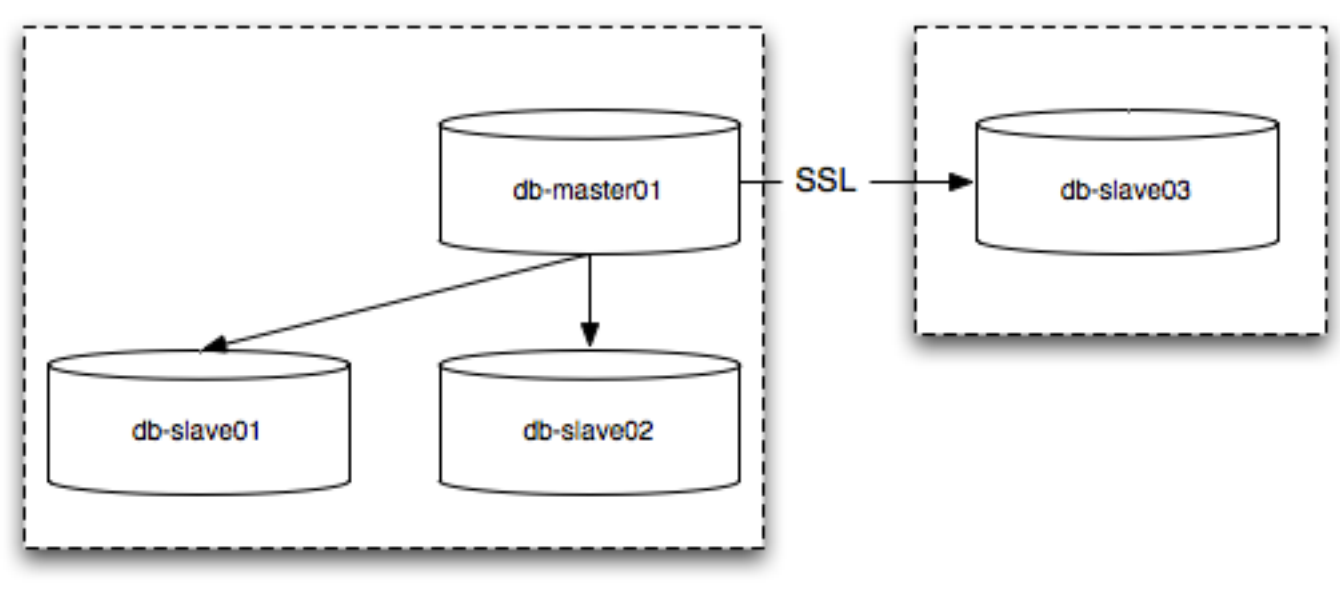
# Backups / Failover

- Relatively easy to setup
  - Optimal to have some DBA knowledge
- Could read the official docs or

  http://wiki.postgresql.org/wiki/Streaming_Replication

# Multiple Standbys

- Can “horizontally scale” your Postgres instances
- Read-Only scale out
  - esp. if reads account for a lot of work



# Case: Business Intelligence

- "Can you find out how many customers are using blue envelopes over the past week and cross reference it against our sales from last year at this time?”
  - “For a report going out today"

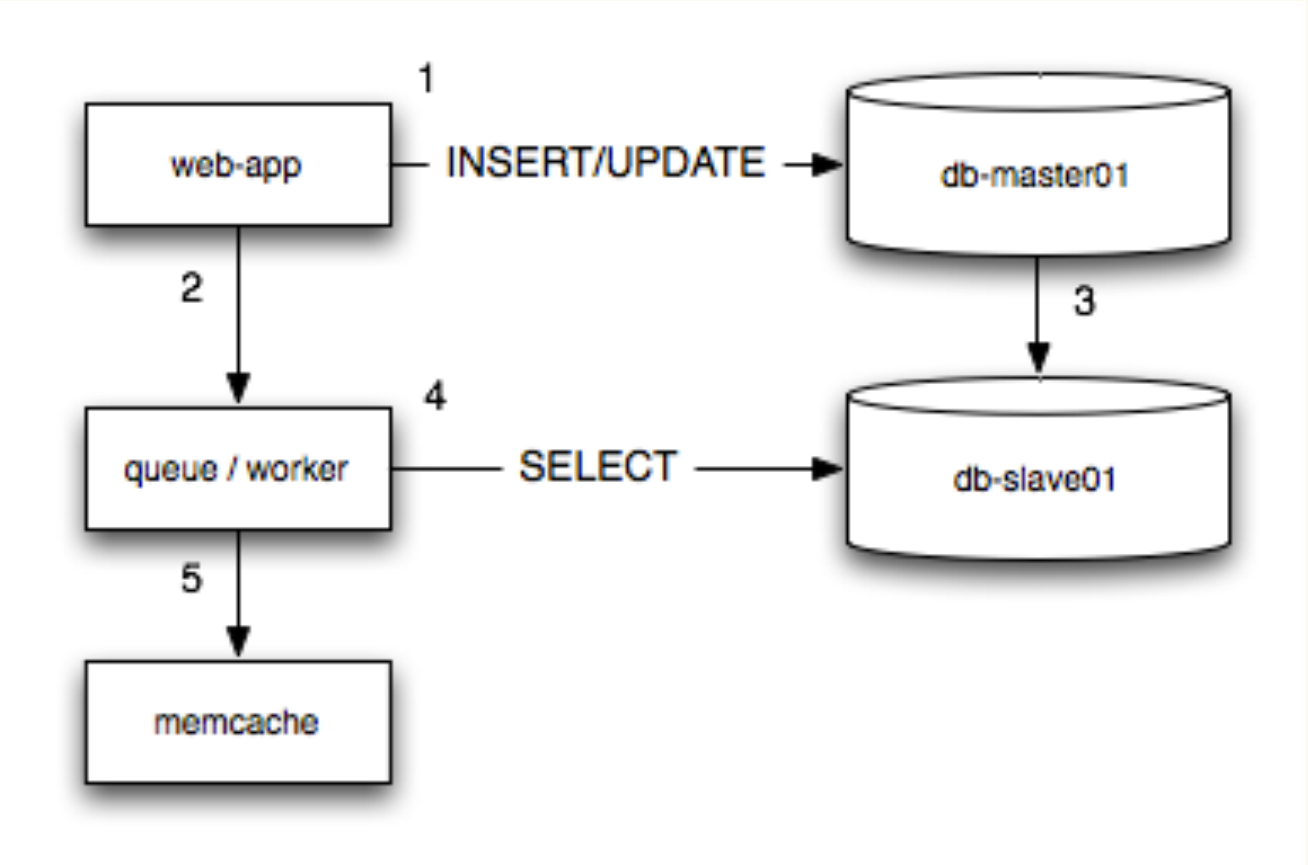
# Solutions

- 8.4
  - Make a SQL dump of tables/database and run query locally
    - time consuming
  - Run the query on the production server
    - Bad user experience, i.e. slow site
- 9.0
  - run the query on a hot standby instance!
  - (Web) application for standby node tailored for business intelligence

# Case: Caching

- 8.4
  - Run a query, cache it's results (memcache, etc.)
- 9.0
  - Can warm up a cache using data from hot standby

# Example


1
web-app
INSERT/UPDATE
db-master01
2
3
4
queue / worker
SELECT
db-slave01
5
memcache

# Case: Changing Master Servers

- (without Slony or other tools)
- 8.4
  - Turn site off, dump data, transfer data, load data, site on
- 9.0
  - New servers acts as hot standby
  - Turn site off, wait for standby to finish catching up, switch, turn site on

# Case: Redirect Read-Only Queries

- Use hot standbys for read only queries
  - Maintenance situations
  - Offload work
    - (Just cache?)
- Caveat emptor: performance may vary

# Case: Full Text Search

- PostgreSQL full text search: tsearch2
- Uses GIN or GiST indexes
- GIN
  - Faster to search over, slower to update
- GiST
  - Slower to search over, faster to update
- (Can't have our strudel and eat it)

# Our Path Deviates Slightly

- Will talk about Sphinx search engine
- …and we will get back to PostgreSQL 9
- …and the web

# Sphinx: High Performance Indexing + Search

- Written in C
- Supports PostgreSQL and some other open source RDBMS
- Makes full text search…fast.  Really fast.

# Back to the Strudel Problem

- Sphinx 0.9 forces you to do a complete reindex when updating search set
  - No problem if data is small or not updated frequently
  - But…
    - Highly dynamic data set
    - Lots of write once, read-only data
- (Sphinx 1.10: incremental indexing! Stay tuned…)

# So: Sphinx or tsearch2?

- Depends on the use-case
- Tools available
  - Ruby has “ThinkingSphinx” library for Ruby ⇔ Sphinx access
- Sphinx is “yet another service”
- Write-once, read many times
  - tsearch2 + GIN and Sphinx both do this well, so…

# Benchmark Battle!

- depesz did a very interesting, elaborate benchmarking
- Source: http://www.depesz.com/index.php/2010/10/17/why-im-not-fan-of-tsearch-2/
- Next few slides use some content from above source

# The Machine

- CPU: Dual core, 2.93GHz Intel Core2Duo E7500
- Memory: 4GB
- Storage: Seagate Barracuda LP – SATA (3Gb/s) – 1TB
- Ran against PostgreSQL 8.4.4
  - (I would expect similar results with 9)

# The Setup

- Used DB of ~19M records
- Broke up into smaller tables for comparison
- Broke up tests by word saturation in text (e.g. 30%, 20%, 5%)

# Setting up the tsearch2 indexes

| table | index type | size | size/records | time | time/records |
|---|---|---|---|---|---|
| pages_1000 | gist | 204,800 B | 204 B | 0.8 s | 0.8 ms |
| pages_1000 | gin | 2,867,200 B | 2,867 B | 1.0 s | 1.0 ms |
| pages_10000 | gist | 2,105,344 B | 210 B | 4.8 s | 0.5 ms |
| pages_10000 | gin | 12,795,904 B | 1,279 B | 5.5 s | 0.6 ms |
| pages_100000 | gist | 27,885,568 B | 278 B | 72.9 s | 0.7 ms |
| pages_100000 | gin | 127,565,824 B | 1,275 B | 82.4 s | 0.8 ms |
| pages_1000000 | gist | 220,954,624 B | 220 B | 659.5 s | 0.7 ms |
| pages_1000000 | gin | 1,057,144,832 B | 1,057 B | 822.9 s | 0.8 ms |
| pages_10000000 | gist | 2,236,841,984 B | 223 B | 9,168.4 s | 0.9 ms |
| pages_10000000 | gin | 10,583,187,456 B | 1,058 B | 88,613.0 s | 8.9 ms |

# tsearch2 and Searching

Order by timestamp, first 20 records; time in milliseconds

| table | index type | ~ 30% word | ~ 20% word | ~ 10% word | ~ 5% word | ~ 1% word | ~ 0.5% word |
|---|---|---|---|---|---|---|---|
| pages_1000 | gist | 612.8 | 649.5 | 538.1 | 509.2 | 442.4 | 408.1 |
| pages_1000 | gin | 3.1 | 3.1 | 2.9 | 2.8 | 2.7 | 2.7 |
| pages_10000 | gist | 3,163.5 | 2,288.3 | 2,395.3 | 2,457.8 | 1,885.3 | 2,747.1 |
| pages_10000 | gin | 8.3 | 7.5 | 4.9 | 5.0 | 3.3 | 3.2 |
| pages_100000 | gist | 48,619.5 | 44,112.9 | 46,061.2 | 41,082.9 | 44,503.1 | 30,890.6 |
| pages_100000 | gin | 38.8 | 32.2 | 24.0 | 14.5 | 7.0 | 5.2 |
| pages_1000000 | gist | 385,316.4 | 380,671.1 | 421,210.0 | 355,074.1 | 276,791.6 | 245,679.6 |
| pages_1000000 | gin | 316.2 | 257.5 | 192.8 | 127.7 | 40.8 | 26.2 |
| pages_10000000 | gist | 6,233,637.7 | 5,390,065.0 | 5,699,556.1 | 4,403,095.1 | 4,152,560.1 | 4,927,288.8 |
| pages_10000000 | gin | 99,674.3 | 88,011.8 | 77,026.5 | 68,944.5 | 280.3 | 263.4 |

# My Interpretation

- GiST gets pwned
- GIN works well, but…
  - Explodes on large table, minus searches for sparse keywords

# Setting up the Sphinx Indexes

| data set | size | size/records | time | time/records |
|---|---|---|---|---|
| **pages_1000** | 1.7 MB | 1657 B | 0.203 s | 0.203 ms |
| **pages_10000** | 8.1 MB | 840 B | 1.119 s | 0.112 ms |
| **pages_100000** | 100 MB | 1040 B | 15.330 s | 0.153 ms |
| **pages_1000000** | 1.1 GB | 1099 B | 189.044 s | 0.189 ms |
| **pages_10000000** | 13 GB | 1354 B | 2580.042 s | 0.258 ms |

# Sphinx and search

Order by timestamp, first 20 records; time in milliseconds

| table | ~ 30% word | ~ 20% word | ~ 10% word | ~ 5% word | ~ 1% word | ~ 0.5% word |
|---|---|---|---|---|---|---|
| pages_1000 | 0.003 | 0.003 | 0.004 | 0.003 | 0.003 | 0.003 |
| pages_10000 | 0.008 | 0.005 | 0.005 | 0.006 | 0.005 | 0.006 |
| pages_100000 | 0.026 | 0.025 | 0.027 | 0.022 | 0.026 | 0.023 |
| pages_1000000 | 0.110 | 0.110 | 0.109 | 0.108 | 0.108 | 0.109 |
| pages_10000000 | 1.139 | 1.140 | 1.147 | 1.140 | 1.140 | 1.145 |

# A Table Says 1,000 words (30% of them)

Comparison for 10 million rows, time in seconds

| index type | ~ 30% word | ~ 20% word | ~ 10% word | ~ 5% word | ~ 1% word | ~ 0.5% word |
|---|---|---|---|---|---|---|
| gist | 6238.225 | 5392.750 | 5700.967 | 4404.192 | 4153.318 | 4926.797 |
| gin | 99.566 | 88.180 | 77.109 | 68.868 | 0.303 | 0.286 |
| sphinx | 1.141 | 1.139 | 1.140 | 1.139 | 1.136 | 1.142 |

- But in sphinx 0.9, there is a time penalty on index creation

# Which Should I Use?

- Up to you – you know your data best
  - Benchmark!
  - Infrastructure setup
    - Access to adding new services
  - What tools are available in your programming language?

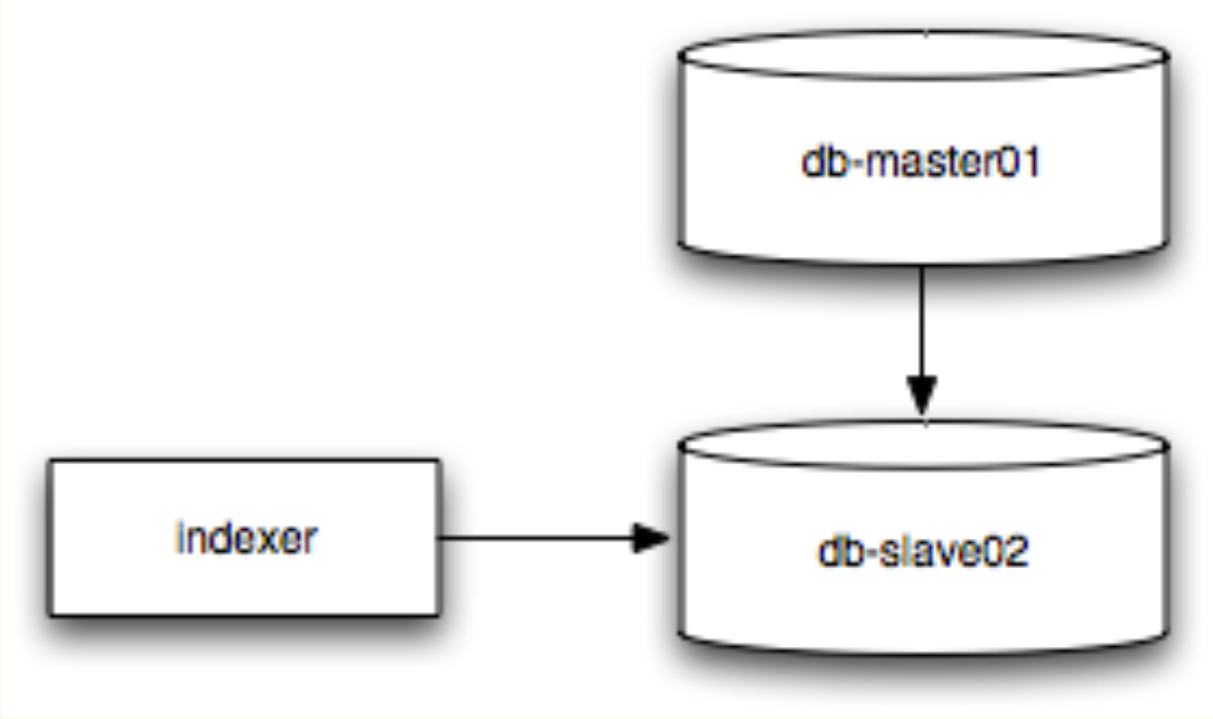
# Why are we discussing this?

- (Other than to fill time)
- I actually had a similar problem:
  - one table has 10 million rows
  - a related table has about 7.5 million
  - both frequently updated
  - both need to be full text searchable

# With PostgreSQL 8.4…

- Could only run Sphinx indexer against master database
  - Some ideas do not need to be attempted

# With PostgreSQL 9.0…




- would keep architecture same for Sphinx 1.10

# Other Notes

- Sphinx libraries
  - http://sphinxsearch.com/community/plugins/

# So…

- PostgreSQL 9 is awesome – spread the word
  - pg_upgrade makes upgrade from 8.4 really easy
  - only issues we've had have been self-inflicted
- Scaling your web infrastructure requires you to understand
  - your data
  - application usage
  - the complexities of communication

# Thank You


Thank You grazie GRACIAS OBRIGADO Tack
THANK YOU DANKE MERCI
December 7th, 2010
Thank you for attending!
Questions?
Contact Info:
jonathan.katz@excoventures.com
THANK YOU DANKE MERCI