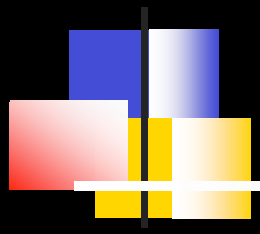
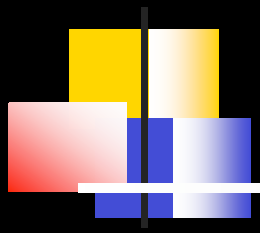
Sistemas Escalables



Club de Investigación Tecnológica
San José, Costa Rica

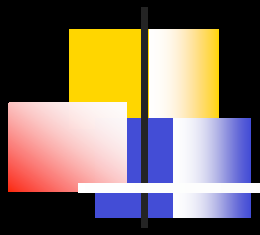
Theodore Hope

22 de septiembre de 2009



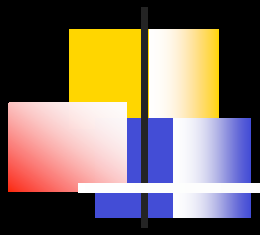
¿Qué es …?

- Escalabilidad
  - La capacidad de dar servicio, con la misma calidad y con más recursos, en la medida que aumenta la carga al sistema.
- Rendimiento (“Performance”)
  - Uso óptimo de recursos para ejecutar una tarea más rápidamente.



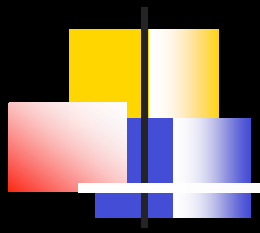
¿Por qué y para qué?

- Si no escalamos, no podemos crecer.
  - Explosión de (inter-)conectividad
  - Web 2.0
- Escalabilidad
  - ⇒ Compromiso con niveles de servicio futuros.
- Rendimiento
  - ⇒ Compromiso con niveles de servicio de hoy.



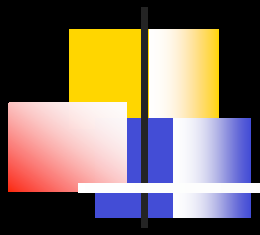
¿Qué queremos escalar?

- Servidores Web
- Servidores de Aplicación
- Servidores de Computación
- Servidores de Base de Datos
- Almacenaje
- Red



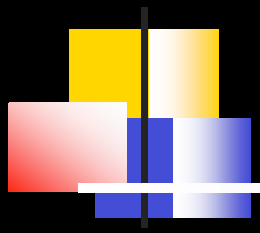
Escalabilidad vs. HA

- Escalabilidad y Alta Disponibilidad (HA) están relacionados.
  - Muchas veces van juntos
  - Pero no son lo mismo
  - Uno no implica el otro (ni vice-versa)



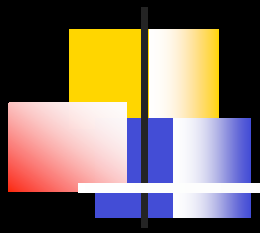
¿Cómo Escalar?

- Escalar Verticalmente (“Scale up”)
  - Reemplazar “el” recurso con otro más poderoso
- Escalar Horizontalmente (“Scale out”)
  - Agregar recursos individuales
- Escalar Diagonalmente
  - Horizontal ⇒ Vertical



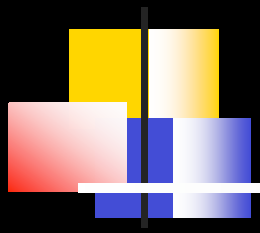
Scale Up

- Caso típico: “un servidor más grande”
  - Fácil de implementar
    - No hay cambio de arquitectura
  - Pero…
    - No escala linealmente en precio/potencia
    - Límite de escalabilidad
    - Generalmente hay un SPOF detrás de esto :-\
    - ¿Cómo calcular la capacidad/potencia del nuevo?



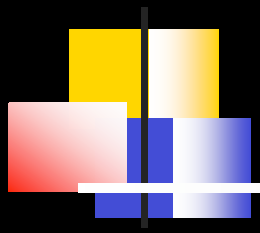
Scale Out - Web, App, Comp

- *N* servidores realizando las mismas tareas.
- Arquitectura que permite agregar, incrementalemente, más servidores.
- Migración inicial complicada, luego fácil ;-)
- Mismo concepto para:
  - Web, Aplicación, Computación
  - BD un poco más complicado



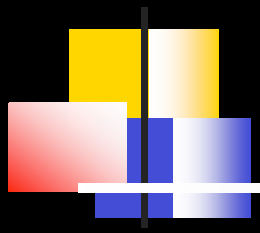
Scale Out - Web, App, Comp

- Load Balancer (LB) distribuye solicitudes entre distintos servidores.
- Funciona en Layer 4 (TCP) o aplicación, e.g.:
  - HTTP, DNS, Oracle, etc.



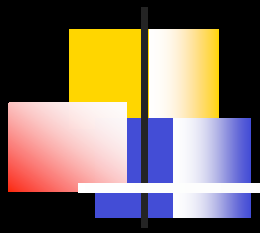
Scale Out - Web, App, Comp

- *N* servidores…
  - Idealmente idénticos
  - Cualquier servidor puede atender cualquier solicitud
- Arquitectura "Shared-Nothing"
  - Servidores independientes y autosuficientes sin punto de contención entre ellos.
  - No se puede usar almacenamiento local
    - Todo debe ir a un DB (o cluster de DB) común



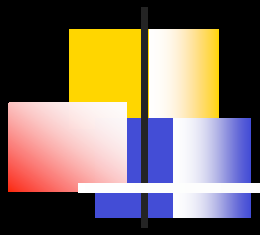
Scale Out - RDBMS

- No es fácil…
- Alternativas a Scale Out para RDBMS:
  - Scale Up (CPU, RAM, I/O)
  - Optimización de queries, schema, etc.
  - Ojo: HA importante, e.g., replicación o "cluster"
- … pero eventualmente necesario.



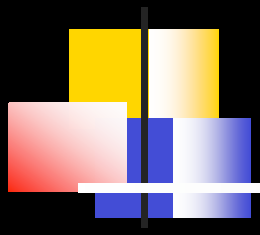
Scale Out - RDBMS

- Escalando para lectura:
  Cuando hay muchos más R que W
  - Un servidor primario de R/W, múltiples servidores secundarios de R/O con réplica
  - Servidores Web/App leen de réplicas R/O y escriben al primario.
  - Problema con retardo de replicación



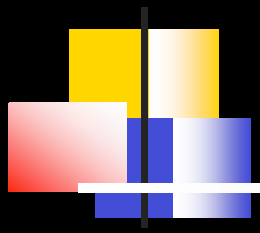
Scale Out - RDBMS

- Escalando R/W
  - Múltiples servidores R/W
    - "Cluster" de dos o más servidores
    - Oracle RAC, MySQL Cluster, etc.
  - Servidores Web/App leen y escriben a cualquier servidor RDBMS.
  - Implica HA



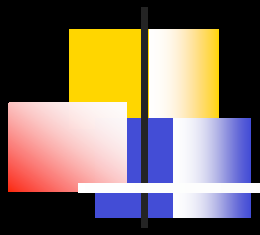
Scale Out - RDBMS

- "Partitioning" / "Sharding"
  - Dividir una BD o tabla en varios pedazos
  - Vertical Partitioning:
    - Migrar ciertas columnas poco usadas a otra tabla
  - Horizontal Partitioning:
    - Agrupar filas en tablas individuales
    - Por rango de llave (e.g., fechas, usuario, etc.)
  - App-level Partitioning:
    - Migrar ciertas tablas a distintos servidores BD



Scale Out - RDBMS

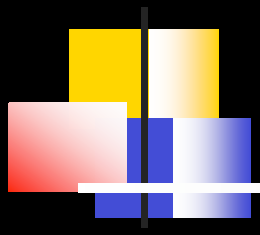
- Desnormalización
  - Redundancia de datos (individual o en grupo)
    - Almacenaje adicional :-(
    - Inconsistencias y falta de integridad referencial
      Arreglarlo es problema del App :-(
    - Buenísimo para lectura; difícil para escritura
  - No lo temamos cuando es en pro de eficiencia
    - “Summary Tables”
    - Usado en OLAP



Scale Out - RDBMS

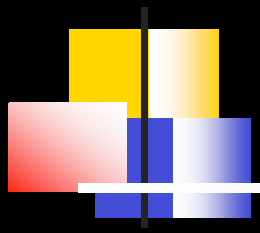
- ¡No todos los datos fueron creados iguales!
  - Ciertos modelos toleran retardos e inconsistencias (“eventual consistency”)
  - Saber para cuáles sí y para cuales no, e.g.,
    - Inconsistencias temporales tolerables:
      catálogo, status updates, noticias, etc.

      versus
    - ACID y HA:
      shopping cart, control de sesiones, pagos, etc.



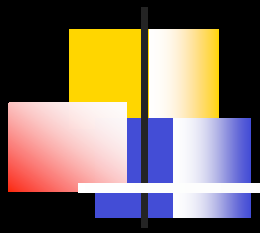
Caching

- Evitar hacer lo mismo una y otra vez…
  - Accesar datos
  - Calcular resultados
  - Generar páginas o partes de páginas
- Implementable en casi todos los niveles:
  - Entre servidor Web y archivos
  - Entre servidor Web y servidor App
  - Entre App y RDBMS



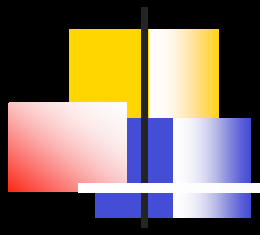
Caching

- Squid (squid-cache.org)
  - Web cache y HTTP proxy
- Memcached (danga.com)
  - Agrega una capa de s/w entre el App y BD
    - "Query" busca primero en cache antes de b.d.
    - Todo en memoria; distribuido y escalable
    - Facebook, Slashdot, SourceForge, Wikipedia, etc.



Escalando en Almacenaje

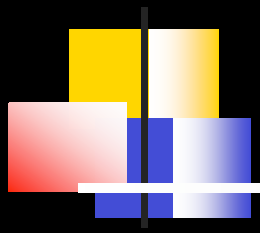
- Sistemas que permitan agregar hardware dinámicante
- Considerar un SAN / NAS
- Usar servicios de Cloud Computing ;-)



Escalando en Red

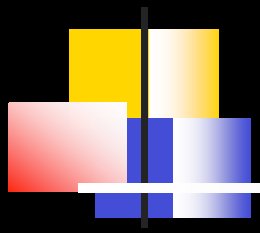
- Vertical:
  - Más ancho de banda sobre el mismo canal
- Horizontal:
  - Agregar canales adicionales

- Para contenido estático, usar un CDN
  - Akamai, Amazon CloudFront



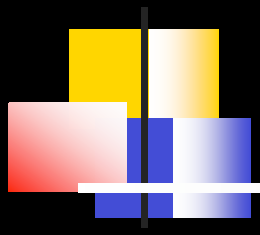
Escalando en Red: Web App

- Minimizar el tráfico entre cliente (browser) y servidor web:
  - Web cache o reverse proxy (e.g., squid)
  - Compresión de html/css/js
  - Headers para caching de contenido en el browser
- Herramientas:
  - YSlow
  - Google Page Speed



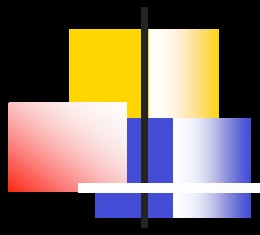
Escalabilidad y CC

- Servicios de Cloud Computing son ideales para implementar arquitecturas escalables
  - Amazon AWS, GoGrid, Mosso, Google App Engine, Windows Azure
- Ojo: Escalabilidad elástica (↑,↓)
  - Carga variable según la época



Límites de Escalabilidad

- Escalabilidad horizontal no es infinita: está limitada por grado de paralelismo.
- Con cada servidor (nodo) adicional hay retorno decreciente.
- Amdahl's Argument (1967):
  - La mejora en el rendimiento debido al aumento en número de procesadores/nodos está limitado por la fracción secuencial (no paralelizable).

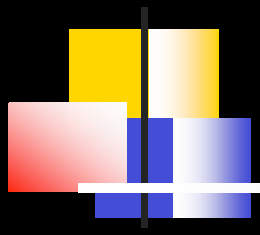


Parafraseando a Amdahl

$$\text{Aumento (“speed-up”)} = \frac{1}{(1-P) + (P/N)}$$

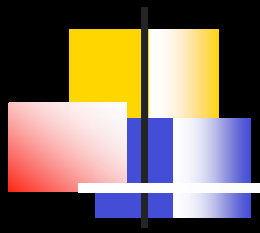
P = fracción de operaciones que son paralelizables
N = número de procesadores

- Ejemplos con N infinito:
  - P = 95%, aumento máx = 20x
  - P = 50%, aumento máx = 2x
- Lecciónes:
  - Minimizar cuellos de botella (secuenciales)
  - Saber qué estamos paralelizando y qué no



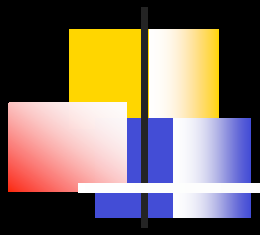
Buenas Prácticas & Consejos

- "Pensamiento Horizontal" a todo nivel:
  - Diseñar para Scale Out desde el inicio
  - "Agregue otro(s)" y no "Uno más grande"
  - "Amperios" en lugar de "Voltios" ;-)
- Evitar tener "*el*" servidor (SPOF)
- Caching (donde tenga sentido)
- Evitar sobrecargar recursos que son difíciles de escalar



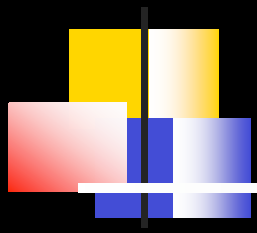
Buenas Prácticas & Consejos

- Desacoplamiento (o Acoplamiento Débil) de componentes
- Uso de SOA con APIs REST entre componentes
- Considerar MTTR para restaurar un servidor
- Colaboración entre “infraestructura” y “dev”



¿Cuándo Escalar?

- Monitoreo y medición de todo, en detalle
  - No sólo CPU, mem, disco, etc.
  - Operaciones del RDBMS, Web serv, Apps
  - Instrumentación a todo nivel
  - Tendencias, puntos de inflexión
  - Herramientas: Nagios, Munin, Cacti, etc.
- Profiling y Benchmarking



¿ ?