Big Data Management.
Sesión 04. Big Data Processing.

Autor: Ziani El Ali, Adil.

adilziani.wordpress.com

5 de mayo de 2020

Índice

Escalabilidad, elasticidad y sus límites

- Un sistema **escalable** tiene la capacidad de añadir recursos y aprovecharlos sin introducir sobrecarga al sistema. Bien añadiendo servers (escalado horizontal) o mejorando los existentes con más recursos (escalado vertical).

- Un sistema **elástico** es un sistema flexible, capaz de agregar o reducir recursos según la necesidad de cada momento. Es una cualidad que se suele asocia a los sistemas Cloud.

Ambas cualidades son un reto en Management, pues siempre buscamos tener/configurar un sistema óptimo para las necesidades de nuestro negocio que es cambiante con el tiempo. Pero en ocasiones a priori desconocemos el consumo real de nuestro sistema, y no existe una definición matemática clara y definitiva que nos permita estimar las necesidades del mismo.

- Una aproximación matemática a la escalabilidad:
  + Universal Scalability Law USL

Universal Scalability Law

- Generalización de Amdahl Law, que modela la escalabilidad de un sistema teniendo en cuenta las fraccionies de carga de trabajo que no se pueden paralelizar (Amdahl Law) y las sobrecarga que se añade al sistema debido a la comunicación entre sí de las partes de ésta.
- Modela la escalabilidad tanto a nivel de Software como hardware.
- **Definición**:

$$C(N) = \frac{N}{1 + \sigma(N-1) + \kappa N(N-1)} \tag{1}$$

  - N: tamaño del sistema. Ej, número de nodos
  - C: rendimiento del sistema. Ej, número consultas/seg, tiempo/tarea fija
  - σ: factor penalización debido a la componente *serial fraction* del *workload*
  - κ: factor penalización debido a la interconectividad entre las partes del sistema.

Cómputo de USL

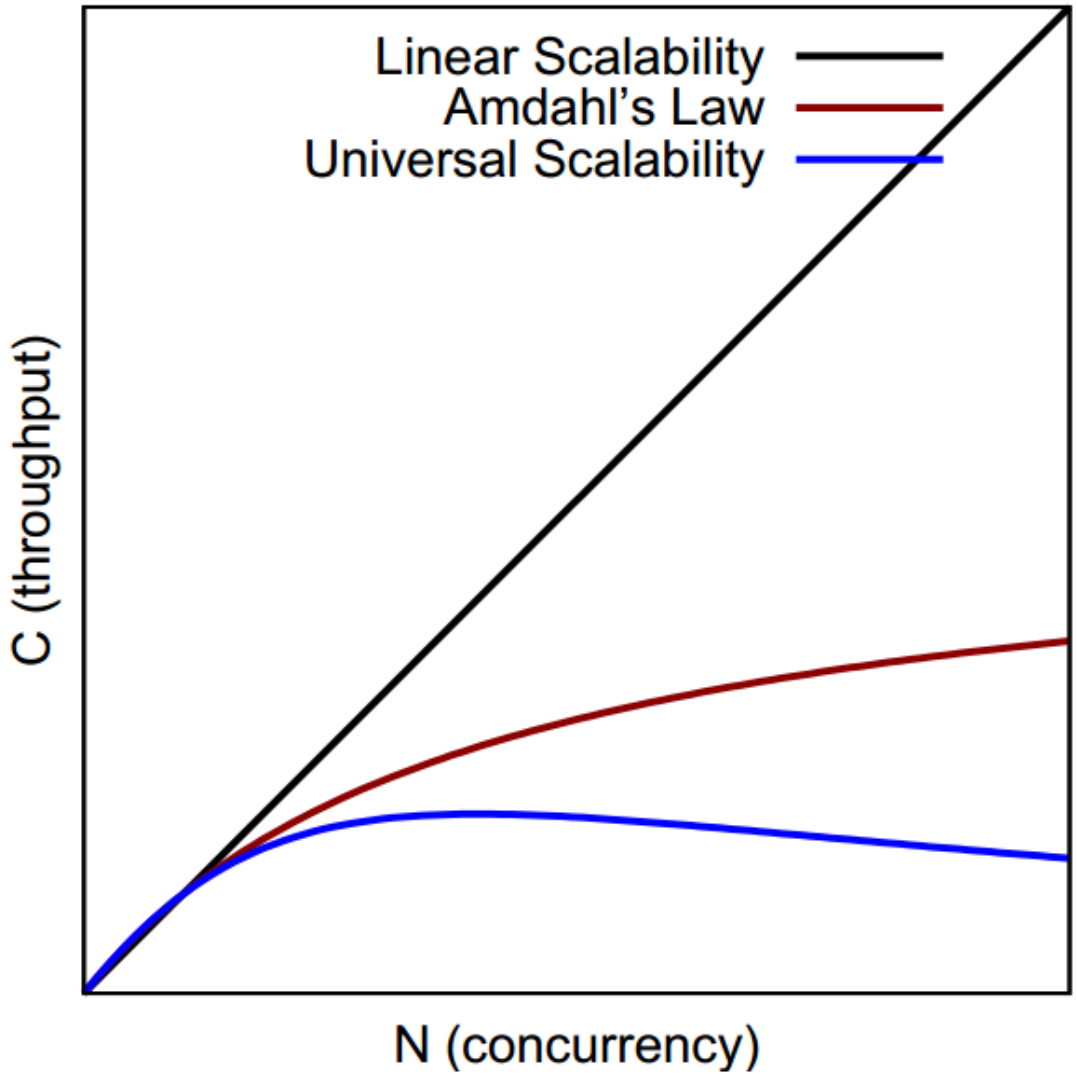
Observad que necesitamos aproximar σ y κ para nuestro sistema.

1. De manera empírica, por ejemplo fijando una tarea pesada, computa $C(N)$ para diferentes valores de N y elabora una tabla de valores.
2. Normalizar los datos respecto al primer valor de N considerado, es decir, $C(N) = \frac{C_N}{C_1}$
3. Ajustar mediante mínimos cuadrados un polinomio de segundo grado $y = ax^2 + bx$ a los datos normalizados. Donde definimos
   - $x := N - 1$
   - $y := \frac{N}{C(N)} - 1$
4. Deshacer el ajuste $C(N) \sim y = \frac{N}{C(N)} - 1$ luego $ax^2 + bx = \frac{N}{C(N)} - 1$ de donde obtenemos que $ax^2 + bx = \sigma x + \kappa x^2 + \kappa x$
   - $\kappa = a$
   - $\sigma = b - a$

R mediante el paquete nls[1] nos permite realizar dicho ajuste y obtener los parámetros σ y κ.

Para más información consulte (Gunther, Subramanyam y Parvu., 2010) donde también hallara una prueba Hadoop terashort beshmark sobre máquinas en AWS.

[1] `https://stat.ethz.ch/R-manual/R-devel/library/stats/html/nls.html`



By Oscar Romero & Alberto Abelló

La escalabilidad normalmente se mide en *speed-up* y *scale-up*

- *speed-up*: fijado un problema, mide el rendimiento al añadir más hardware. *Speed-up* lineal quiere decir que el mismo problema se resuelve en fracciones de tiempo proporcionales $\frac{T}{N}$
- *scale-up*: mide el rendimiento del sistema con problemas de tamaño variable. *scale-up* lineal quiere decir que N máquinas resuelven $N * T$ problemas en el mismo tiempo.

USL nos muestra que un escalado lineal no
es posible, y por tanto se ha de considerar maximizar la función $C(N)$. En la realidad no buscamos el máximo teórico si no también minimizar el coste de la infraestroctura.

Moraleja: más nodos a nuestro Cluster no necesariamente equivale a mejor rendimiento, existe un límite e incluso podría degradar el sistema. Se busca un equilibrio.

Desafíos en los procesamientos distribuidos

En la sesión 01 vimos algunos desafíos en Big Data como Velocidad, Variedad, Volumen etc. Ahora tenemos nuestro Cluster distribuido y datos distribuidos. Qué desafíos/problemas se generan?

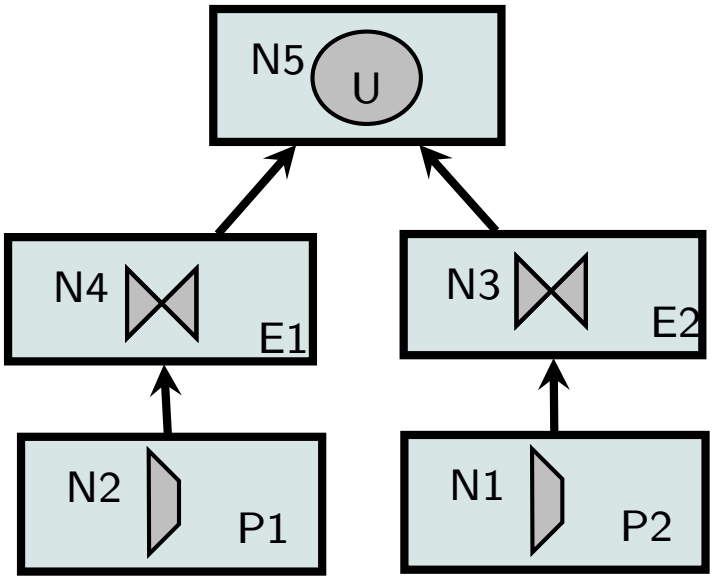
- Diseño de bases de datos distribuidas
  - Fragmentación y particionado.
  - Replicación.
  - Número de nodos a considerar de los disponibles.
- Catálogo distribuido
  - Catálogo global vs catálogo local.
  - Centralizado en un nodo, o distribuido?
  - Una copia vs multi-copia.
- Quieries distribuidas.
  - Cómo están distribuidos los datos para generar un mejor acces plan.
  - Sobrecarga debido a la comunicación entre nodos.
- Seguridad
  - Cómo fortalecer la seguridad en la red y la comunicación entre nodos, asegurar los datos que intercambian entre ellos

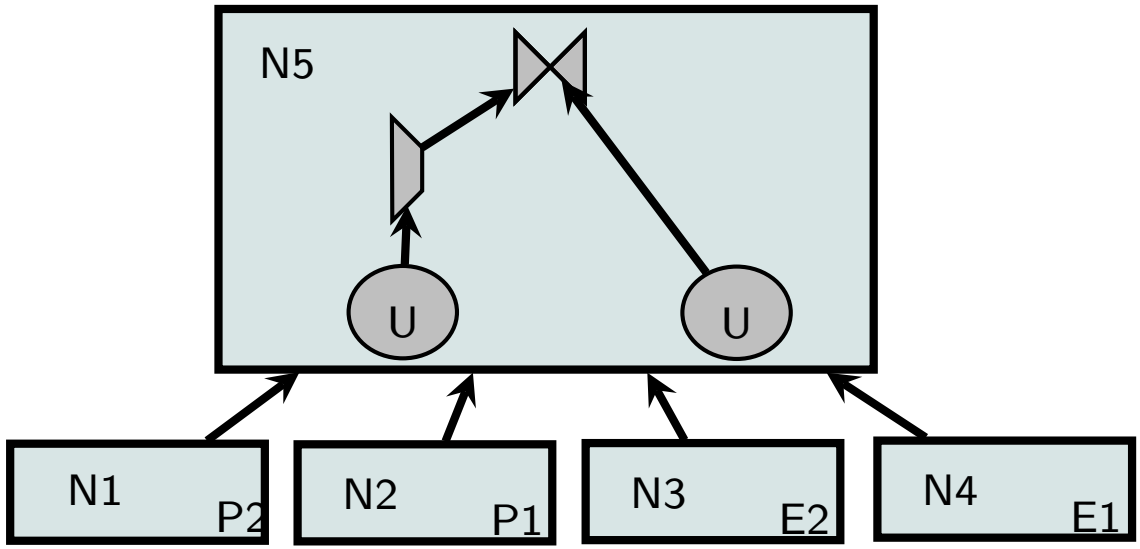
Todas estas cuestiones, cada plataforma Big data y cada base de datos NoSQL intenta responder de alguna manera. Debemos conocer nuestros interés para escoger la herramienta que más se adapte a nosotros.

Actividad

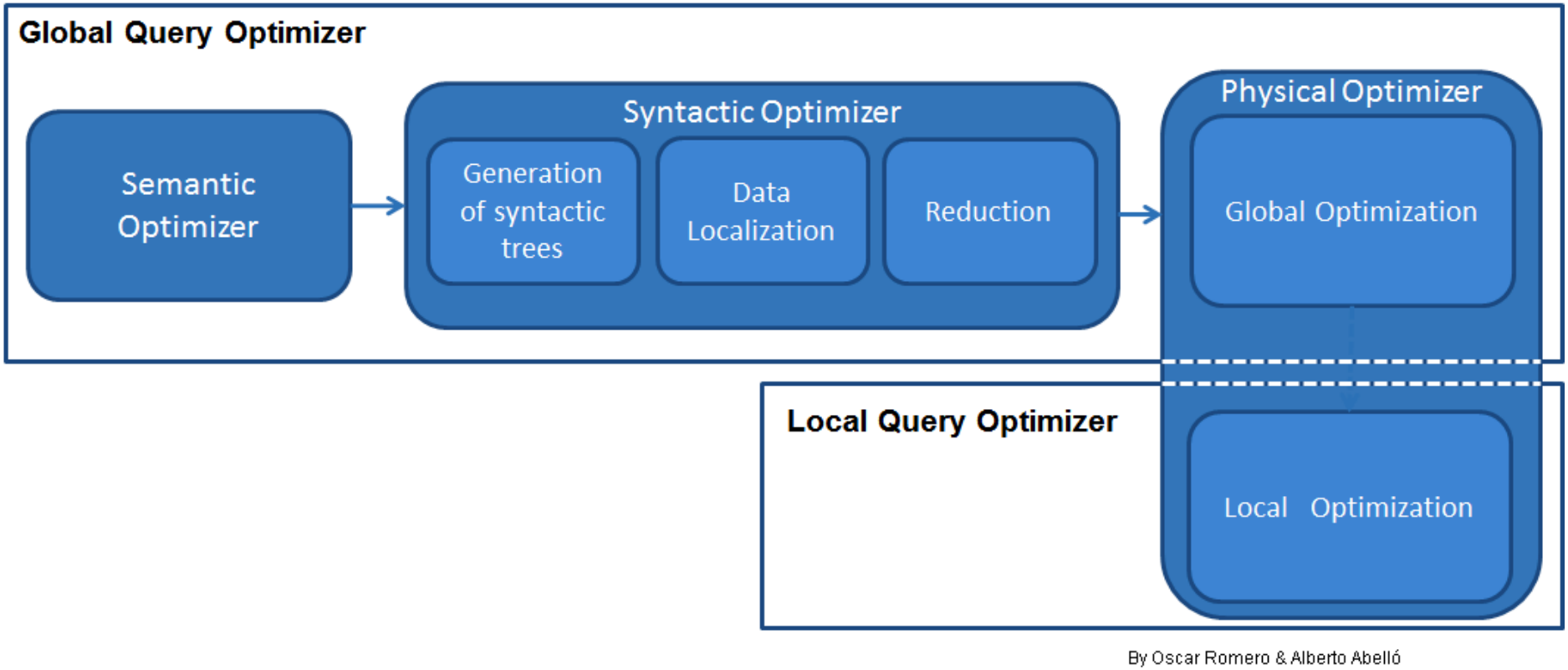
- Consideramos que tenemos un Cluster con 5 slaves: N1, N2, N3, N4, N5.
- Dos bases de datos distribuidas: Empleados y Puestos.
- Para la siguiente consulta, qué plan de acceso considera mejor?, por qué?

```
SELECT * FROM Empleados e, Puestos p
WHERE e.id_empleado = p.id_empleado
AND p.puesto = 'Data_Architect'
```





Fases de una query distribuida



By Oscar Romero & Alberto Abelló

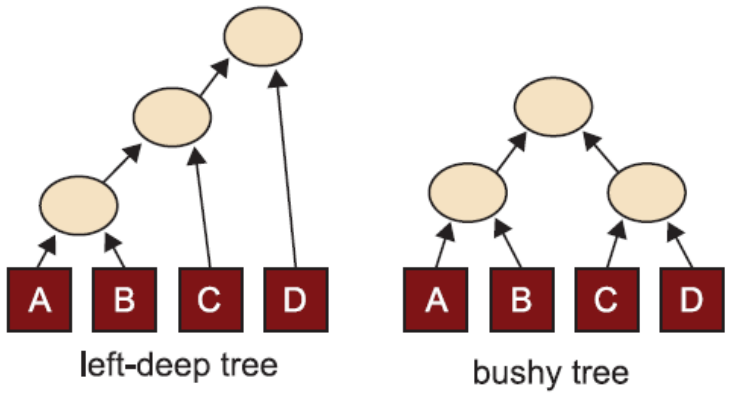
Pysical Optimizer

- Fase donde se transforma una consulta interna en un plan de acceso eficiente.
  - Remplazar las operaciones algebraicas por los correspondientes algoritmos.
  - Decidir el orden de ejecución de éstos
    - Enumerar las alternativas y generar distintos diagramas de ejecución
    - Estimar costes
    - Buscar la mejor solución, haciendo uso de estadísticas y situación actual del sistema

Planes de ejecución alternativos

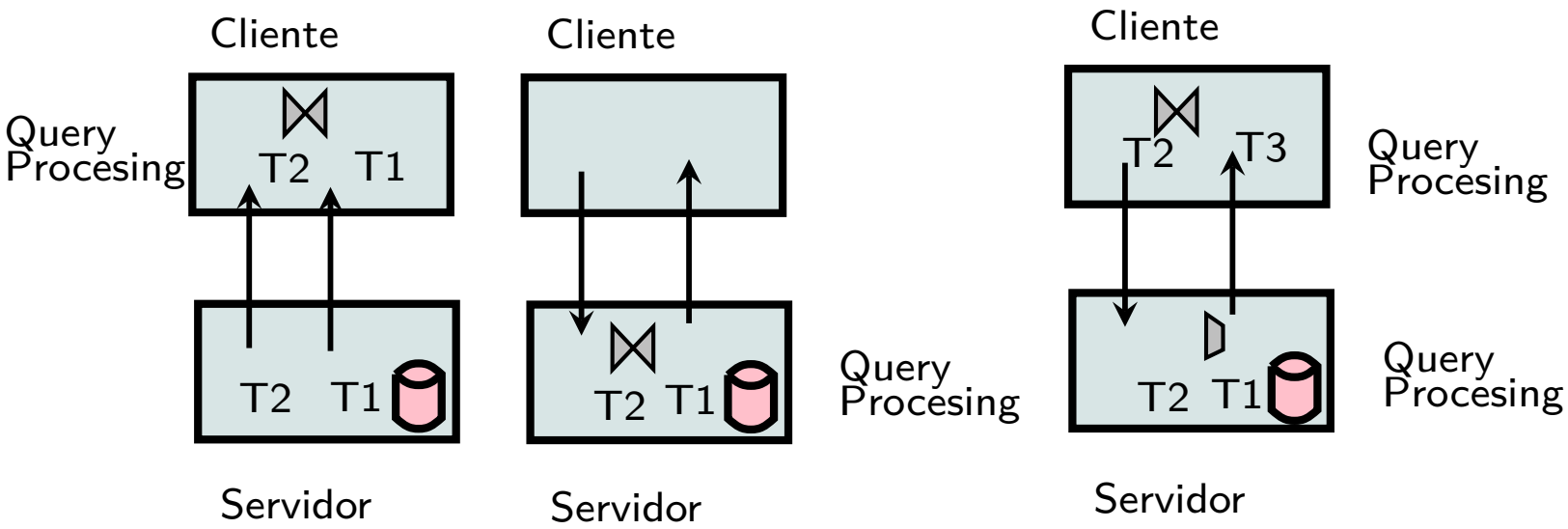
Aspectos que se tienen en cuenta:

- Orden workflow
  - *Left or right deep trees*
  - *Bushy trees*
- Selección de partición.
  De qué nodo tomar los datos
  - Comparar coste de joins internos
- Algoritmos
  - Cuales explotan más el paralelismo



Data shipping, Query shipping

- *Data shipping*
  - Los datos son recuperados de la máquina donde se almacenan hacia la máquina cliente que ejecuta la consulta.
    - Evitar cuellos de botella en los datos de uso frecuente.
    - Evitar sobrecarga sobre la máquina que almacena los datos en caso de tener diversos solicitudes de clientes.
- *Query Shiping*
  - La consulta se delega a la máquina que posee los datos, ésta envía el resultado al cliente.
    - Reduce la latencia de red al mandar menos datos (si las consultas son selectivas)
    - Puede generar cuellos de botella si recibe demasiadas solicitudes.
- *Hybrid Architecture*
  - Combina ambas arquitecturas, para datos muy frecuentes *Data shipping*, para datos selectivos *Query shipping*



Tipos de paralelismo

- *Inter-query parallelism*
  - Diversas consultas no conflictivas se ejecutan en paralelo en múltiples procesos para mejorar el rendimiento global.
  - Común en el procesamiento de transacciones en línea (OLTP), donde varios usuarios concurrentes envían solicitudes al sistema.

- *Intra-query parallelism*
  - Una única consulta es distribuida en distintos *jobs* para se ejecutada por múltiples procesadores.
  - Más frecuente en sistemas distribuidos.
  - Dos tipos:
    + *Intra-operator*
    - *Static or dynamic partitioning:* ejecutar la consulta en paralelo sobre cada partición. Unir resultados.
    + *Inter-operator*
    - Pipeline: se ejecutan las partes de una consulta una tras otro. Ejemplo, selección y agregación.

Criterios para escoger un plan de acceso

- Tiempo de respuesta
  - Tiempo necesario para ejecutar la consulta.
  - Paralelismo. Dividir la consulta en N operaciones.
- Coste total del modelo
  - Coste local en cada máquina + coste comunicación.
    - Coste por unidad de procesamiento.
    - Coste I/O.
    - Coste iniciar y enviar mensaje entre máquinas.
    - Coste de mandar un Byte por la red.
    - Tamaño de la unidad elemental de procesamiento.
    - Estimar operaciones intermedias.
- Considerar soluciones híbridas en cuanto a arquitecturas de paralelismo.

Ejemplo cálculo coste

- Parámetros
  - Procesamiento local
    - Tiempo medio de CPU para procesar una instancia (T_{CPU})
    - Número de instancias a procesar ($\#ins$)
    - Tiempo medio por cada operación I/O ($T_{I/O}$)
    - Cantidad de procesos I/O ($\#I/O$)
  - Procesamiento global
    - Tiempo medio iniciar y enviar mensaje (T_{msg})
    - Número mensajes ($\#msgs$)
    - Tiempo transferencia 1 packet por la red (T_{tr})
    - Número packets ($\#packet$)
- Cálculo
  - Recursos $= T_{CPU} * \#inst + T_{I/O} * \#I/Os + T_{msg} * \#msgs + T_{tr} * \#buckets$

En la próxima sesión

Sesion 05: **Big Data Design.**

- Tipos de arquitecturas NoSQL
- Data Modeling
- Data Storage
- ...

referencias

Gunther, N. J., S. Subramanyam y S. Parvu. (2010). *Hidden Scalability Gotchas in Memcached and Friends.*. http://www.perfdynamics.com/Manifesto/USLscalability.html.