Modelos de descomposición de tareas paralelas

Sistemas Distribuidos

Dra. Cristina Abad

Contenido

- Conceptos de descomposición paralela
- Descomposición paralela basada en tareas
- Descomposición paralela basada en datos
- Algunos conceptos importantes
  - Algoritmos vergonzosamente paralelos
  - Dividir-y-conquistar
- Caso de estudio: MapReduce

Dos maneras de paralelizar algoritmos

1. Paralelizar algoritmos seriales (**re-formulación**)
2. Usar algoritmos totalmente diferentes (**re-diseño**)

En este capítulo nos enfocamos en **formulaciones paralelas**; la meta es discutir cómo se puede re-formular un algoritmo serial, para su ejecución en paralelo

Pasos para construir un algoritmo paralelo

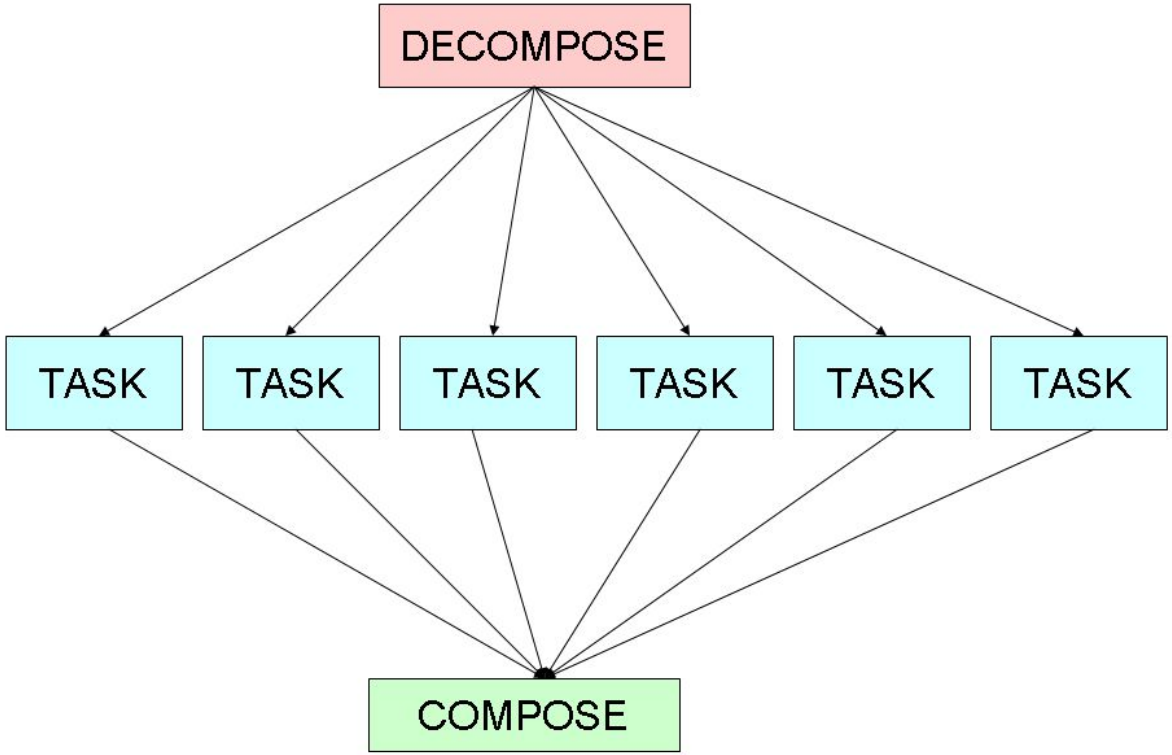
1. Identificar partes del trabajo que se pueden realizar concurrentemente
2. Mapear las porciones concurrentes del trabajo en múltiples procesos que se ejecutan en paralelo
3. Distribuir los datos de entrada, de salida e intermedios, del programa
4. Administrar el acceso a los datos compartidos: evitar conflictos
5. Sincronizar los procesos en etapas de la ejecución del programa paralelo

Metas

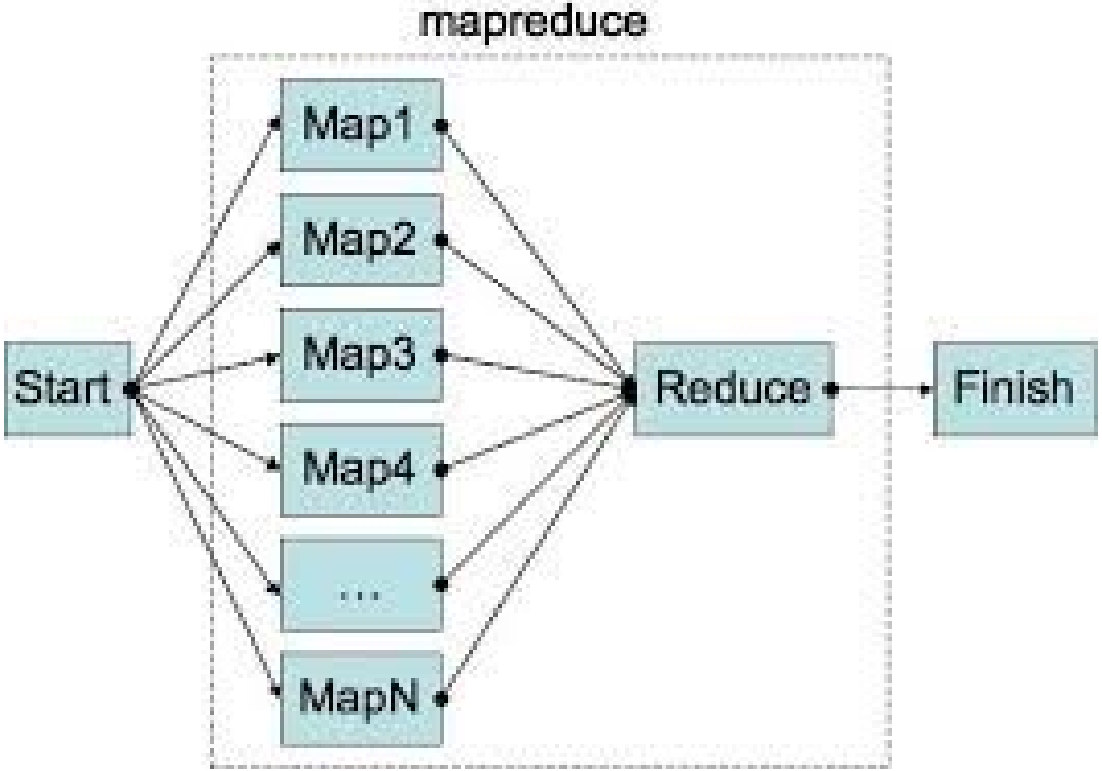
- Maximizar concurrencia
- Reducir sobrecargas (overheads) introducidas por la paralelización
- Maximizar el potencial de aceleración (**speedup**)

Lectura recomendada: Disk-locality in Datacenter Computing Considered Irrelevant: http://people.eecs.berkeley.edu/~ganesha/disk-irrelevant_hotos2011.pdf

Descomposición de tareas



Ejemplo: MapReduce



Conceptos

- Descomposición
  - Dividir un cómputo en partes más pequeñas, las cuales se pueden ejecutar concurrentemente
  - Proceso de dividir el cómputo en pedazos de trabajo más pequeños (tareas)
- Tareas
  - Unidades de computación definidas por el programador
  - Se consideran indivisibles

Ejemplo 1: Multiplicación de matrices densas

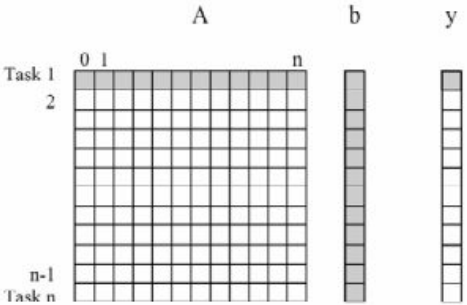


Figure 3.1 Decomposition of dense matrix-vector multiplication into n tasks, where n is the number of rows in the matrix. The portions of the matrix and the input and output vectors accessed by Task 1 are highlighted.

Tasks can be of different size.

- *granularity of a task*

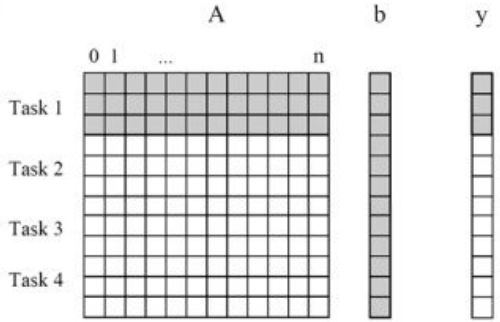


Figure 3.4 Decomposition of dense matrix-vector multiplication into four tasks. The portions of the matrix and the input and output vectors accessed by Task 1 are highlighted.

Algunas observaciones:

- El tamaño de las tareas es uniforme
- No hay dependencias entre tareas
- Todas las tareas necesitan una copia de *b*

Ejemplo 2: Procesamiento de queries

Ejecutar la consulta:

Model =“civic” AND Year = “2001” AND (Color = “green” OR Color = “white”)

en la siguiente base de datos

ID#	Model	Year	Color	Dealer	Price
4523	Civic	2002	Blue	MN	$18,000
3476	Corolla	1999	White	IL	$15,000
7623	Camry	2001	Green	NY	$21,000
9834	Prius	2001	Green	CA	$18,000
6734	Civic	2001	White	OR	$17,000
5342	Altima	2001	Green	FL	$19,000
3845	Maxima	2001	Blue	NY	$22,000
8354	Accord	2000	Green	VT	$18,000
4395	Civic	2001	Red	CA	$17,000
7352	Civic	2002	Red	WA	$18,000

Ejemplo 2: Continuación

Hallando tareas concurrentes:

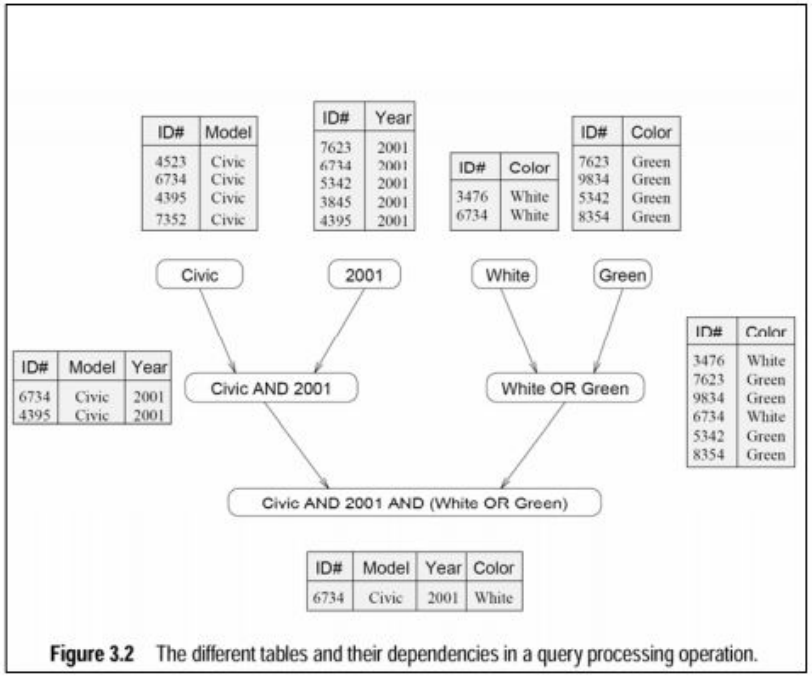


Figure 3.2 The different tables and their dependencies in a query processing operation.

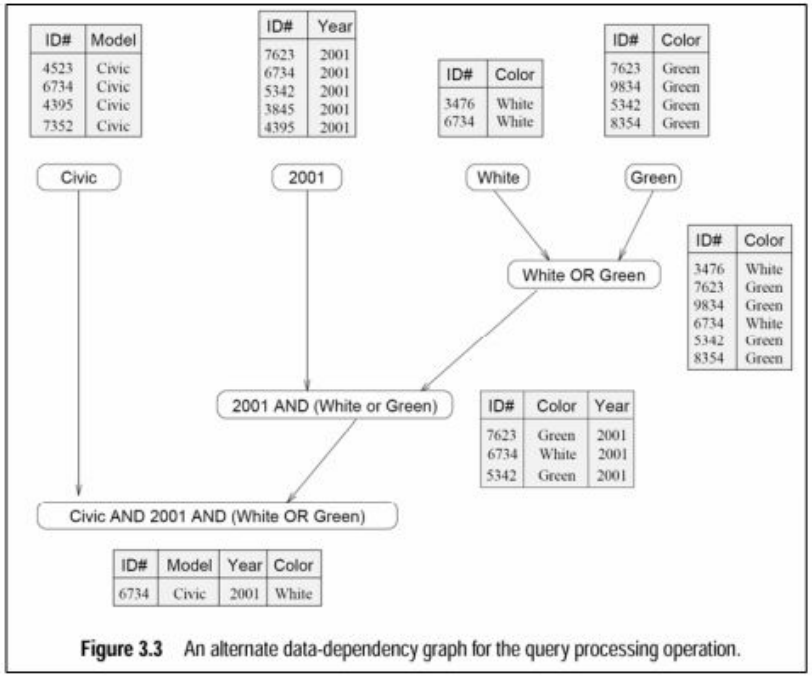


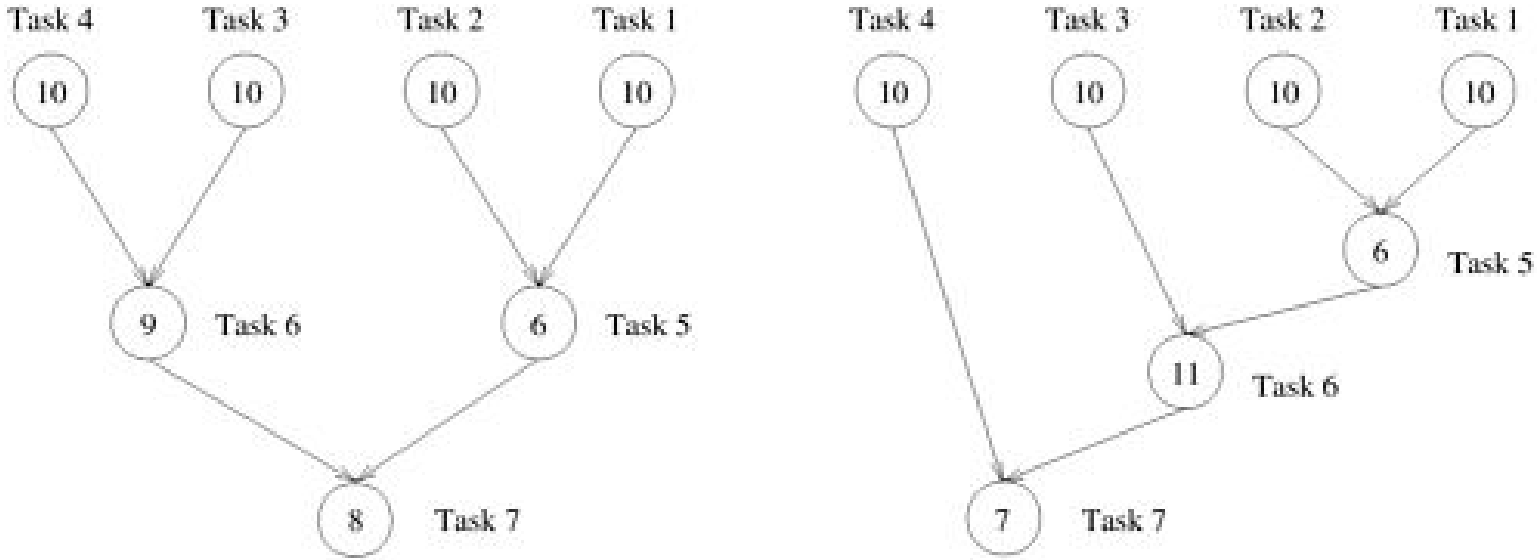
Figure 3.3 An alternate data-dependency graph for the query processing operation.

Grafos de dependencias de tareas

- En la mayoría de los casos, existen dependencias entre las diferentes tareas
  - Algunas tareas no pueden empezar hasta que otras tareas terminen
  - Ej.: productor-consumidor
- Las dependencias se representan usando grafos dirigidos acíclicos (DAGs) llamados **grafos de dependencias de tareas**

Grafos de dependencias de tareas (cont.)

- **Nodo:** representa una tarea
- **Eje dirigido:** representa dependencia de control

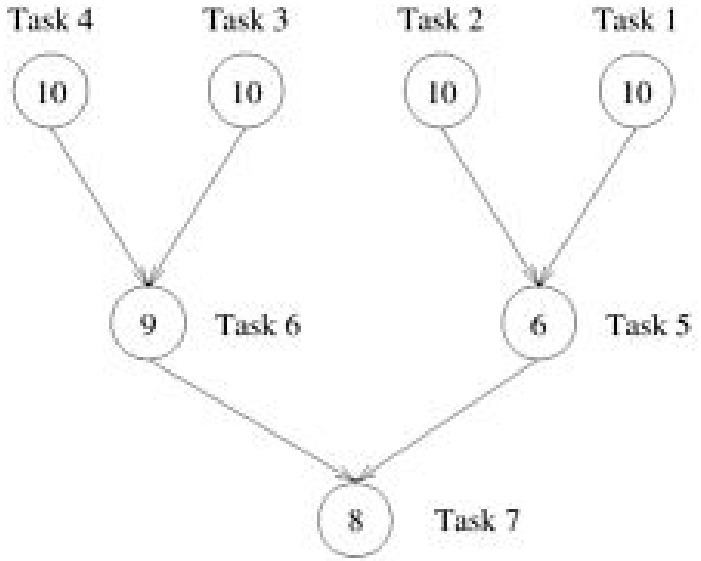


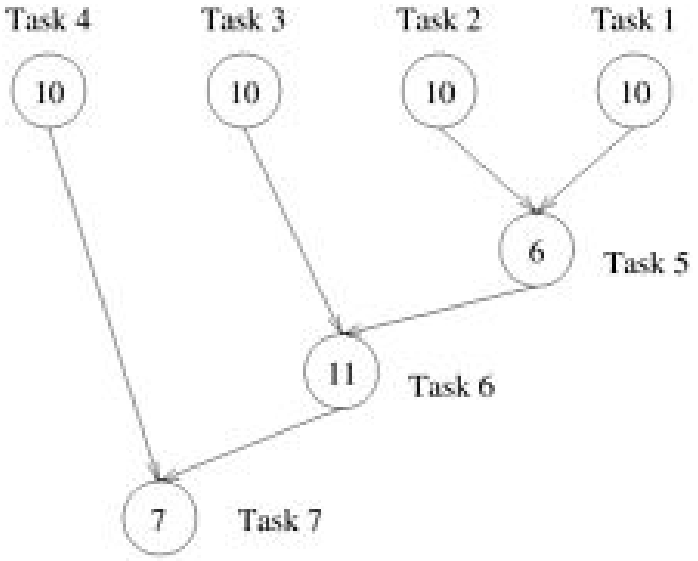
Grafos de dependencias de tareas (cont.)

- **Grado de concurrencia**
  - # de tareas que se pueden ejecutar concurrentemente
  - Generalmente, nos interesa el grado de concurrencia **promedio**
- ¿Grado de concurrencia vs. Granularidad de tareas?
  - Son inversamente proporcionales
- Longitud de la **ruta crítica**
  - La ruta más larga en el grafo (considerando los pesos de los vértices)
    - Los pesos representan el tamaño de las tareas
- La granularidad de las tareas afecta las dos características listadas arriba

Ejercicio

Calcule la **longitud de la ruta crítica** y el **grado de concurrencia** promedio de las dos paralizaciones mostradas abajo.

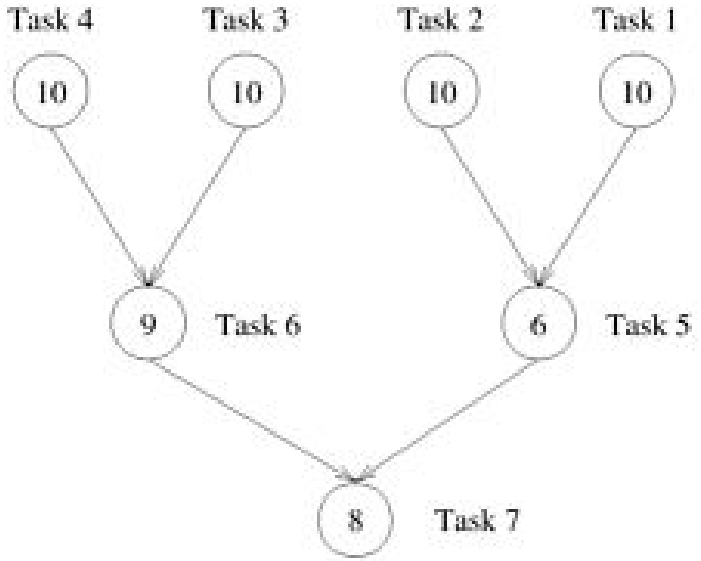


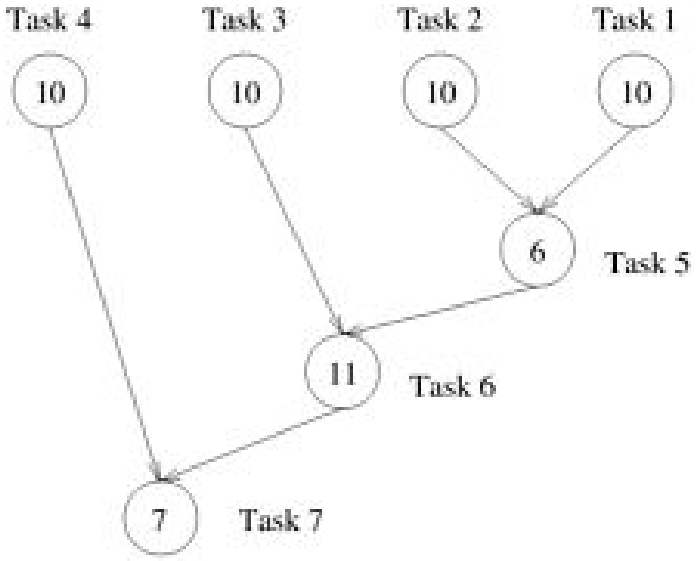


Ejercicio (cont.)

Longitud ruta crítica (izq): 27
Longitud ruta crítica (der): 34
Grado de concurrencia promedio (izq): 7/3 = 2.33
Grado de concurrencia promedio (der): 7/4 = 1.75

Calcule la **longitud de la ruta crítica** y el **grado de concurrencia** promedio de las dos paralizaciones mostradas abajo.

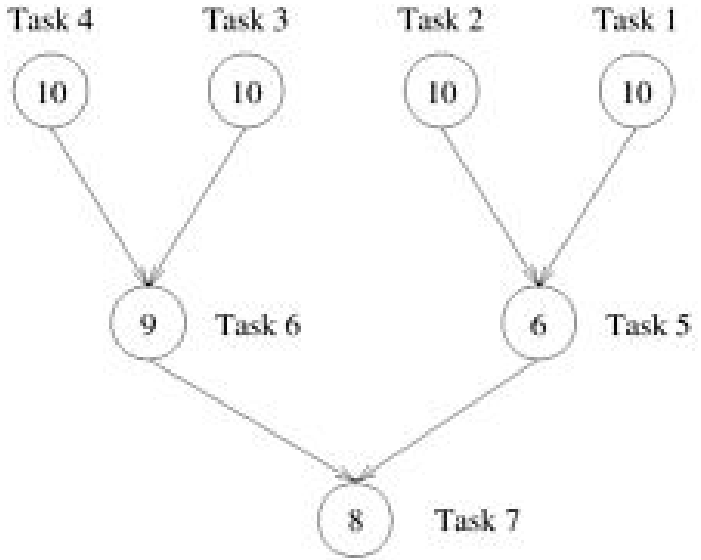


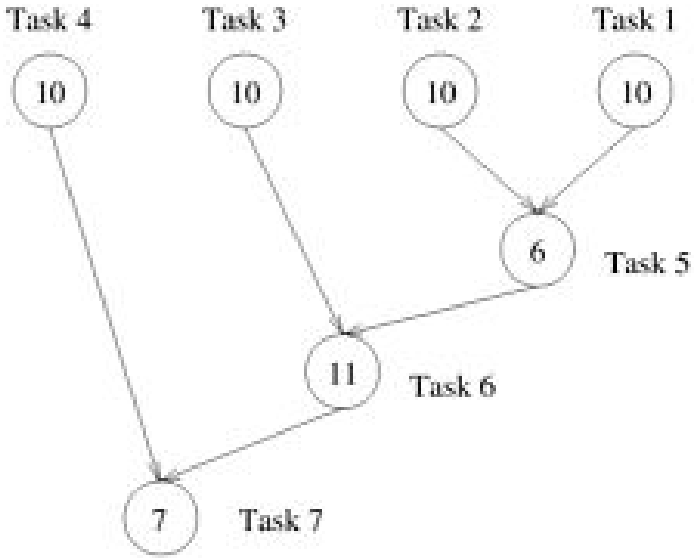


Ejercicio (cont.)

Calcule: **grado de concurrencia promedio, considerando *peso* de las tareas**

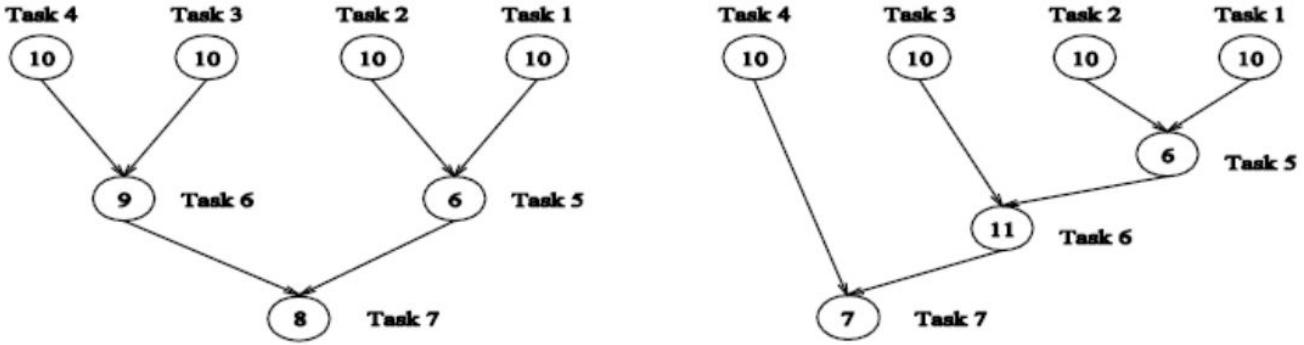
Cantidad de trabajo total / Longitud de la ruta crítica





Solución

Task-dependency graphs of query processing operation



Left graph:

Critical path length = 27

Average degree of concurrency = 63/27 = 2.33

Right graph:

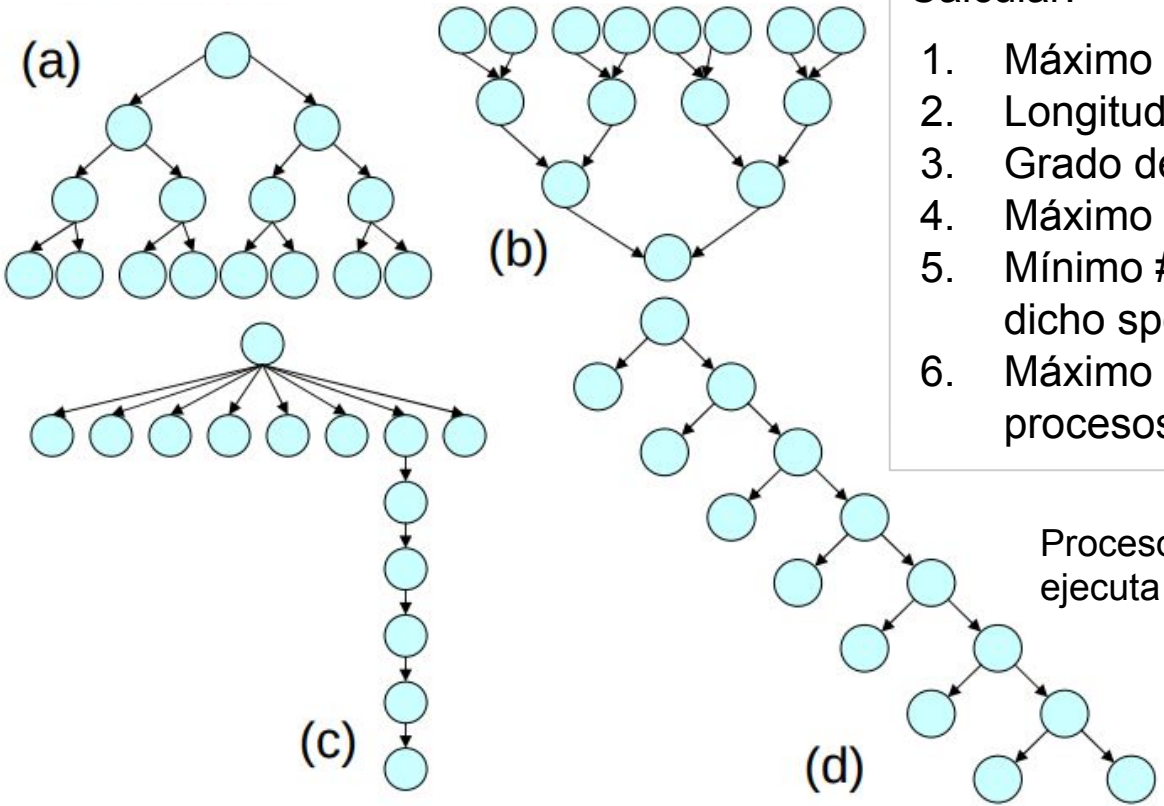
Critical path length = 34

Average degree of concurrency = 64/34 = 1.88

Otras métricas de rendimiento

- Speedup = Tiempo de ejecución secuencial / tiempo de ejecución en paralelo
- Eficiencia paralela = Tiempo de ejecución secuencial / (tiempo de ejecución en paralelo x # de procesos usados)

Ejercicio



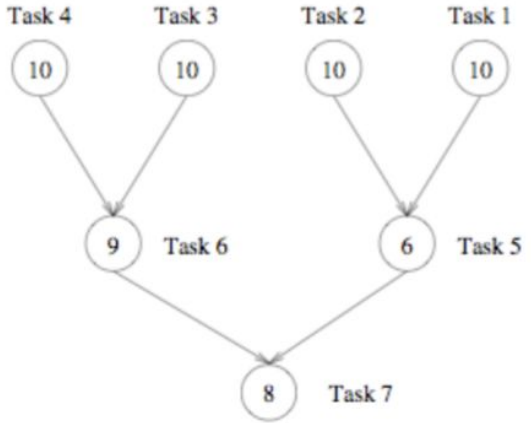
Calcular:

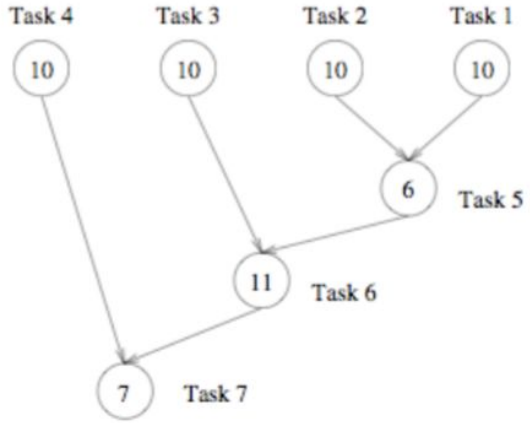
1. Máximo grado de concurrencia
2. Longitud de la ruta crítica
3. Grado de concurrencia promedio
4. Máximo *speedup*
5. Mínimo # de procesos para alcanzar dicho speedup
6. Máximo speedup si limitamos procesos a 2, 4 y 8

Proceso → Agente de procesamiento que ejecuta la tarea; <> a definición de S.O.

Speedup para el ejercicio del query

- Calculate:
  - Maximum concurrency (4)
  - Critical path length (27&34)
  - Average concurrency (2.33 & 1.88)
  - Minimum execution time (cpl)
  - Maximum speedup (ac)
  - # processors for max speedup (mc)
  - Speedup with 2 processors (1.7, 1.68)





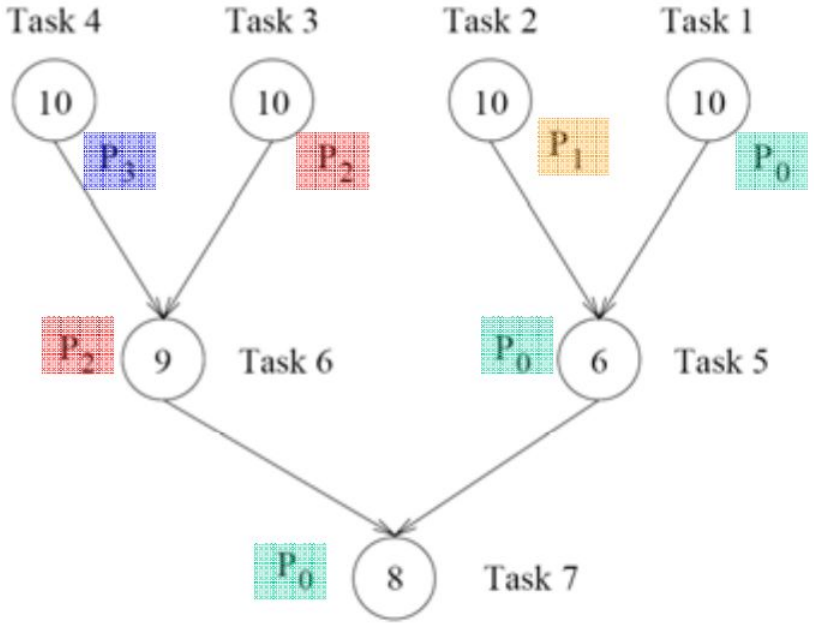
Mapeo de tareas a procesos

- Parte de la *planificación* (*scheduling*)
- Mapeo → asignación de tareas a procesos
- El API no siempre permite controlar el mapeo de tareas a procesos
- Planificación de tareas, procesos e hilos no se puede controlar
- ¿Qué caracteriza a un buen mapeo?

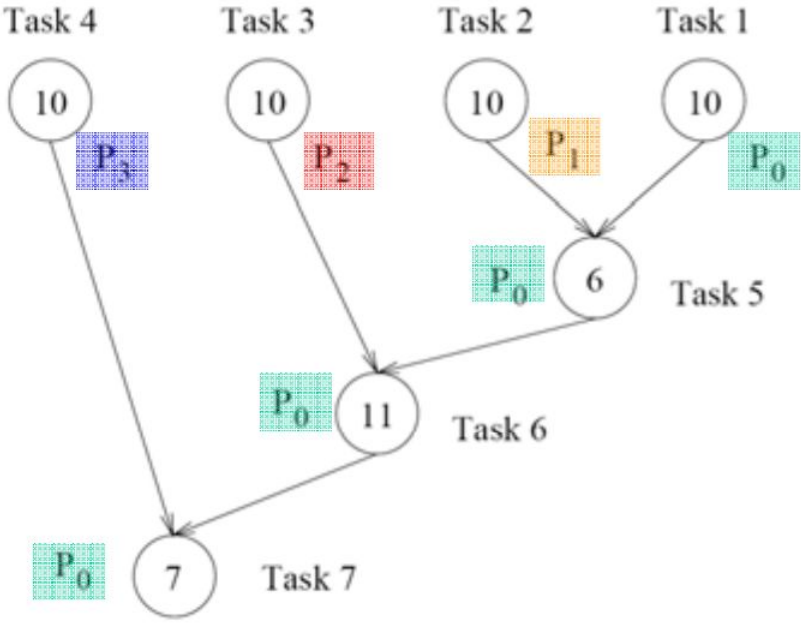
Características de un buen mapeo

- Maximizar concurrencia para mapear tareas independientes a diferentes procesos
- Minimizar interacción al mapear tareas que interactúan en el mismo proceso
- Las metas anteriores pueden estar en conflicto
- Descomposición determina grado de concurrencia
- Mapeo determina cuánta concurrencia se usa y cuán efectivamente
- Otra meta: Minimizar duración total del trabajo, al asegurarse que los procesos están disponibles para ejecutar las tareas en la ruta crítica tan pronto se puedan ejecutar

Ejemplo: Mapeo de tareas a procesos



(a)



(b)

Hadoop map task list for job_200904110811_0003 on ip-10-250-110-47

Completed Tasks

Task	Complete	Status	Start Time	Finish Time	Errors	Counters
task_200904110811_0003_m_000043	100.00%	hdfs://ip-10-250-110-47.ec2.internal/user/root/input/ncdc/all/1949.gz:0+220338475	11-Apr-2009 09:00:06	11-Apr-2009 09:01:25 (1mins, 18sec)		10
task_200904110811_0003_m_000044	100.00%	Detected possibly corrupt record: see logs.	11-Apr-2009 09:00:06	11-Apr-2009 09:01:28 (1mins, 21sec)		11
task_200904110811_0003_m_000045	100.00%	hdfs://ip-10-250-110-47.ec2.internal/user/root/input/ncdc/all/1970.gz:0+208374610	11-Apr-2009 09:00:06	11-Apr-2009 09:01:28 (1mins, 21sec)		10

Figure 5-3. Screenshot of the tasks page

Job job_200904110811_0003

All Task Attempts

Task Attempts	Machine	Status	Progress	Start Time	Finish Time	Errors	Task Logs	Counters	Actions
attempt_200904110811_0003_m_000044_0	/default-rack/ip-10-250-163-143.ec2.internal	SUCCEEDED	100.00%	11-Apr-2009 09:00:06	11-Apr-2009 09:01:25 (1mins, 19sec)		Last 4KB Last 8KB All	11	

Input Split Locations

/default-rack/10.250.202.127
/default-rack/10.250.123.223
/default-rack/10.250.115.79

Go back to the job
Go back to JobTracker

Hadoop, 2009.

FAILED REDUCE attempts in job_1451941227556_0002

Show 20 entries Search:

Attempt	Rack	Status	Node	Logs	Start Time	Shuffle Finish Time	Merge Finish Time	Finish Time	Elapsed Time Shuffle	Elapsed Time Merge	Elapsed Time Reduce	Elapsed Time	Note
attempt_1451941227556_0002_r_000000_0	/default-rack	FAILED	/fd0/ud0/nodemode0.logilevitest-ramsan.b9.internal.cloudapp.net:30060	logs	Wed Jan 6 01:45:14 +0000 2016	Wed Jan 6 01:45:21 +0000 2016	Wed Jan 6 01:45:21 +0000 2016	Wed Jan 6 01:45:21 +0000 2016	6sec	0sec	0sec	6sec	Error: java.lang.RuntimeException: Error in configuring object at org.apache.hadoop.util.ReflectionUtils.setJobConf (ReflectionUtils.java:109) at org.apache.hadoop.util.ReflectionUtils.setConf (ReflectionUtils.java:75) at org.apache.hadoop.util.ReflectionUtils.newInstance (ReflectionUtils.java:133) at org.apache.hadoop.mapred.ReduceTask.runOldReducer (ReduceTask.java:409) at org.apache.hadoop.mapred.ReduceTask.run (ReduceTask.java:392) at org.apache.hadoop.mapred.YarnChild$2.run (YarnChild.java:167) at java.security.AccessController.doPrivileged(Native Method) at javax.security.auth.Subject.doAs (Subject.java:415) at org.apache.hadoop.security.UserGroupInformation.doAs (UserGroupInformation.java:1671) at org.apache.hadoop.mapred.YarnChild.main (YarnChild.java:162) Caused by: java.lang.reflect.InvocationTargetException at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method) at sun.reflect.NativeMethodAccessorImpl.invoke (NativeMethodAccessorImpl.java:57) at sun.reflect.DelegatingMethodAccessorImpl.invoke (DelegatingMethodAccessorImpl.java:43) at java.lang.reflect.Method.invoke(Method.java:606) at org.apache.hadoop.util.ReflectionUtils.setJobConf (ReflectionUtils.java:106) ... 9 more Caused by: java.lang.RuntimeException: configuration exception at org.apache.hadoop.streaming.PipeMapRed.configure (PipeMapRed.java:222) at org.apache.hadoop.streaming.PipeReducer.configure (PipeReducer.java:67) ... 14 more Caused by: java.io.IOException: Cannot run program "ve.exe": CreateProcess error=2, The system cannot find the file specified at java.lang.ProcessBuilder.start (ProcessBuilder.java:1041) at org.apache.hadoop.streaming.PipeMapRed.configure (PipeMapRed.java:209) ... 15 more Caused by: java.io.IOException: CreateProcess error=2, The system cannot find the file specified at java.lang.ProcessImpl.create(Native Method) at java.lang.ProcessImpl.<init>(ProcessImpl.java:385) at java.lang.ProcessImpl.start(ProcessImpl.java:136) at java.lang.ProcessBuilder.start (ProcessBuilder.java:1022) ... 16 more
attempt_1451941227556_0002_r_000000_1	/default-rack	FAILED	/fd0/ud0/nodemode0.logilevitest-ramsan.b9.internal.cloudapp.net:30060	logs	Wed Jan 6 01:45:23 +0000 2016	Wed Jan 6 01:45:29 +0000 2016	Wed Jan 6 01:45:29 +0000 2016	Wed Jan 6 01:45:29 +0000 2016	6sec	0sec	0sec	6sec	Error: java.lang.RuntimeException: Error in configuring object at org.apache.hadoop.util.ReflectionUtils.setJobConf (ReflectionUtils.java:109) at org.apache.hadoop.util.ReflectionUtils.setConf (ReflectionUtils.java:75) at org.apache.hadoop.util.ReflectionUtils.newInstance (ReflectionUtils.java:133) at org.apache.hadoop.mapred.ReduceTask.runOldReducer (ReduceTask.java:409) at org.apache.hadoop.mapred.ReduceTask.run (ReduceTask.java:392) at org.apache.hadoop.mapred.YarnChild$2.run (YarnChild.java:167) at java.security.AccessController.doPrivileged(Native Method) at javax.security.auth.Subject.doAs (Subject.java:415) at org.apache.hadoop.security.UserGroupInformation.doAs (UserGroupInformation.java:1671) at

Métodos de descomposición

- Descomposición basada en tareas
  - Descomposición recursiva
  - Descomposición exploratoria
  - Descomposición especulativa
- Descomposición basada en datos
- Descomposición híbrida

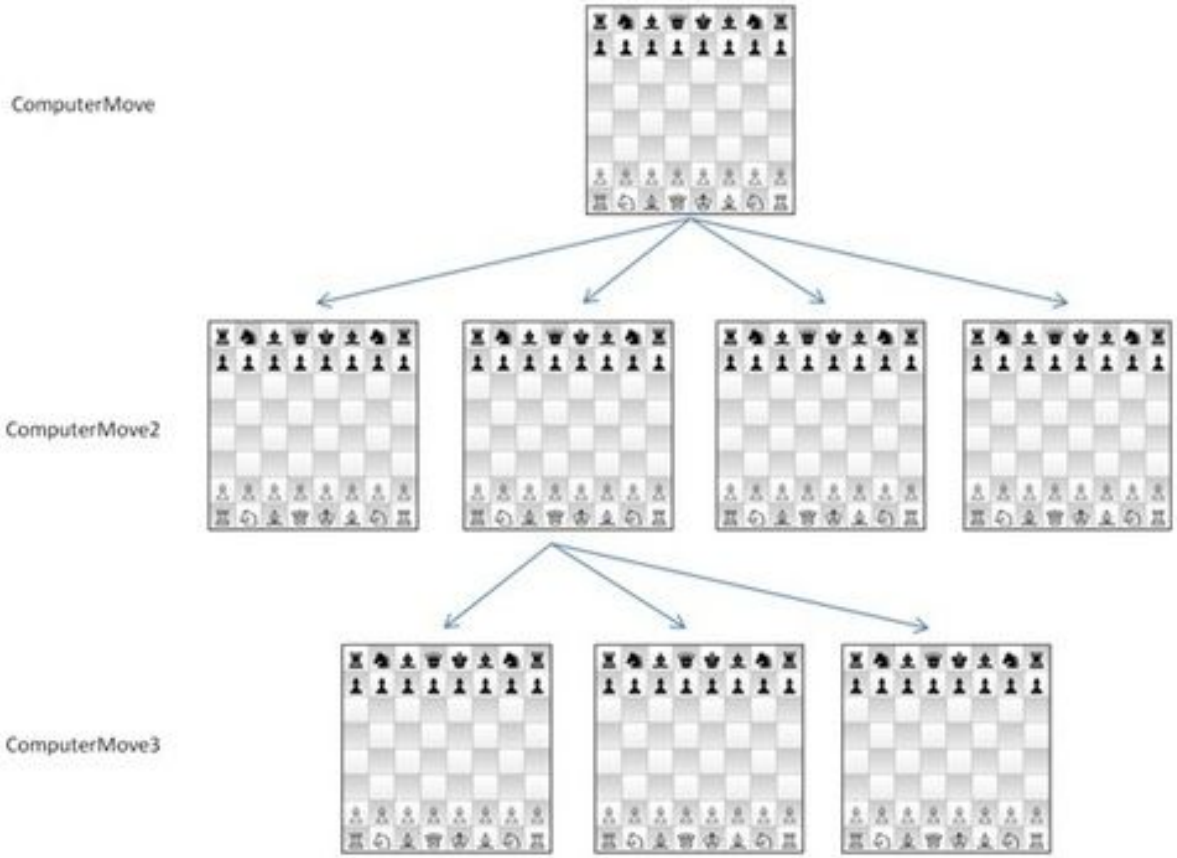
Descomposición recursiva

- Problema que se puede resolver al dividir-y-conquistar
  - Decomponer en subproblemas
    - Hacerlo recursivamente
  - Combinar las sub-soluciones
    - Hacerlo recursivamente
- Concurrencia: los sub-problemas se resuelven el paralelo

Descomposición exploratoria

- Descomposición de problema, de la mano con su ejecución
- Estos problemas típicamente involucran la exploración (búsqueda) de un espacio de estados de soluciones
- Problemas en esta clase incluyen:
  - Varios problemas de optimización discreta
  - Pruebas de teoremas
  - Juegos
- Posible anomalía: Trabajo depende de orden de búsqueda (terminamos cuando encontramos la solución)

Ejemplo



Descomposición especulativa

(no lo vamos a estudiar)

- Dependencias entre tareas no se conocen a-priori
- ¿Cómo identificar las dependencias entre las tareas?
  - Enfoque conservador: identificar tareas que están garantizadas que son independientes
  - Enfoque optimista: planificar tareas, aún si no sabemos si son independientes; se puede hacer un roll-back más adelante
    - → Especulativo
    - Similar al concepto de “branch prediction” en procesadores
- Ejemplo: Simulación paralela de eventos discretos

Descomposición basada en datos

- Ideal para problemas que trabajan con Grandes Datos
- Pasos:
  - Se particionan los datos
  - Partición de los datos se usa para inducir el particionamiento del cómputo en tareas
- Los datos que se particionan puede ser de: entrada, salida o intermedios

Descomposición basada en datos de salida

- Sub-salidas independientes
- Ejemplo: Multiplicación de matrices

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \cdot \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} \rightarrow \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

(a)

$$\begin{aligned} C_{1,1} &= A_{1,1}B_{1,1} + A_{1,2}B_{2,1} \\ C_{1,2} &= A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \\ C_{2,1} &= A_{2,1}B_{1,1} + A_{2,2}B_{2,1} \\ C_{2,2} &= A_{2,1}B_{1,2} + A_{2,2}B_{2,2} \end{aligned}$$

(b)

Figure 3.10 (a) Partitioning of input and output matrices into 2×2 submatrices.

Decomposition I	Decomposition II
Task 1: $C_{1,1} = A_{1,1}B_{1,1}$	Task 1: $C_{1,1} = A_{1,1}B_{1,1}$
Task 2: $C_{1,1} = C_{1,1} + A_{1,2}B_{2,1}$	Task 2: $C_{1,1} = C_{1,1} + A_{1,2}B_{2,1}$
Task 3: $C_{1,2} = A_{1,1}B_{1,2}$	Task 3: $C_{1,2} = A_{1,2}B_{2,2}$
Task 4: $C_{1,2} = C_{1,2} + A_{1,2}B_{2,2}$	Task 4: $C_{1,2} = C_{1,2} + A_{1,1}B_{1,2}$
Task 5: $C_{2,1} = A_{2,1}B_{1,1}$	Task 5: $C_{2,1} = A_{2,2}B_{2,1}$
Task 6: $C_{2,1} = C_{2,1} + A_{2,2}B_{2,1}$	Task 6: $C_{2,1} = C_{2,1} + A_{2,1}B_{1,1}$
Task 7: $C_{2,2} = A_{2,1}B_{1,2}$	Task 7: $C_{2,2} = A_{2,1}B_{1,2}$
Task 8: $C_{2,2} = C_{2,2} + A_{2,2}B_{2,2}$	Task 8: $C_{2,2} = C_{2,2} + A_{2,2}B_{2,2}$

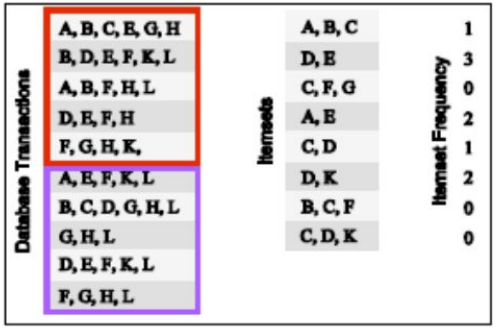
Figure 3.11 Two examples of decomposition of matrix multiplication into eight tasks.

Descomposición basada en datos de entrada

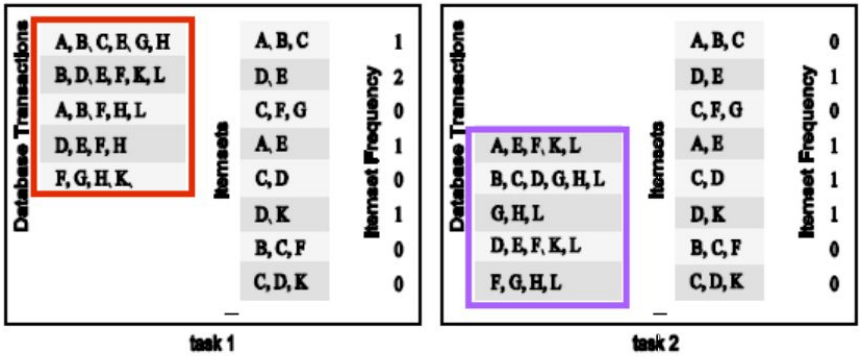
- Aplicable cuando cada salida puede ser computada como una función de la entrada
- En muchos casos, es la única descomposición natural ya que la salida no se conoce a-priori
  - Ej.: encontrar el valor mínimo en una lista, ordenar una lista, etc.
- Una tarea se asocia con cada partición de los datos de entrada
  - La tarea ejecuta lo más que puede del cómputo, con los datos que le fueron asignados
  - Procesamiento subsiguiente **combina** los resultados parciales
- Particionamiento de datos de entrada similar a dividir-y-conquistar

Ejemplo

Count the frequency of itemsets in database transactions



Partition computation by partitioning the set of transactions



Descomposición de datos intermedios

- Muchas veces resulta útil ver a un cómputo como una secuencia de transformaciones de los datos de entrada hasta obtener los de salida
- En estos casos, es beneficiosos usar las fases intermedias como elemento de descomposición
- Particionamiento puede ser basado en la entrada o la salida de una fase intermedia

Regla “el dueño calcula”

- ¿Cómo podemos ir de las descomposiciones de los datos a los mapeos de las tareas?
  - Regla “el dueño calcula”
- Cada tarea “es dueña” de los datos sobre los que calcula y es responsable de realizar todos los cómputos asociados con los mismos
- Descomposición de datos de entrada:
  - Tarea realiza todos los cálculos que se pueden realizar sobre la partición de entrada que le fue asignada
- Descomposición de datos de salida:
  - Tarea realiza todos los cálculos necesarios para obtener la partición de salida que le fue asignada

Descomposición híbrida

En ocasiones, se pueden mezclar los enfoques estudiados anteriormente.

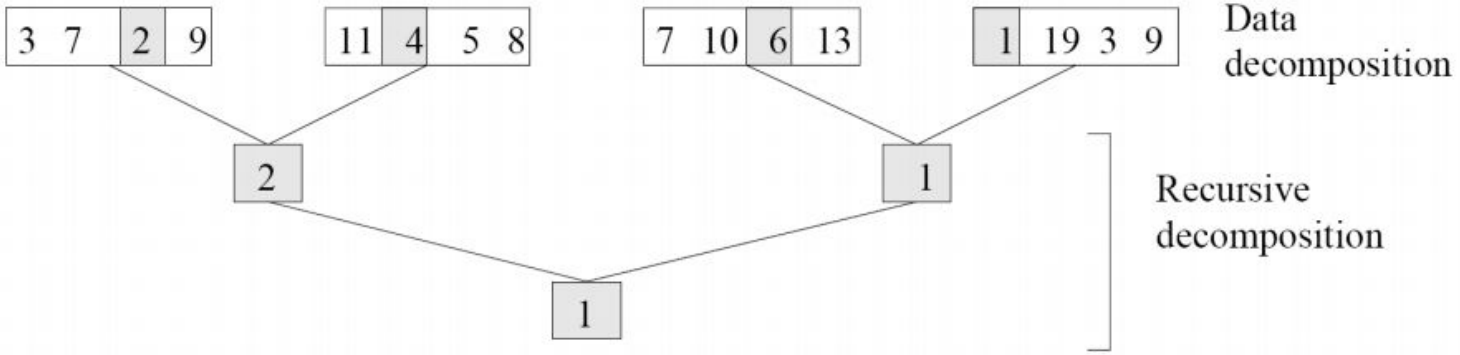
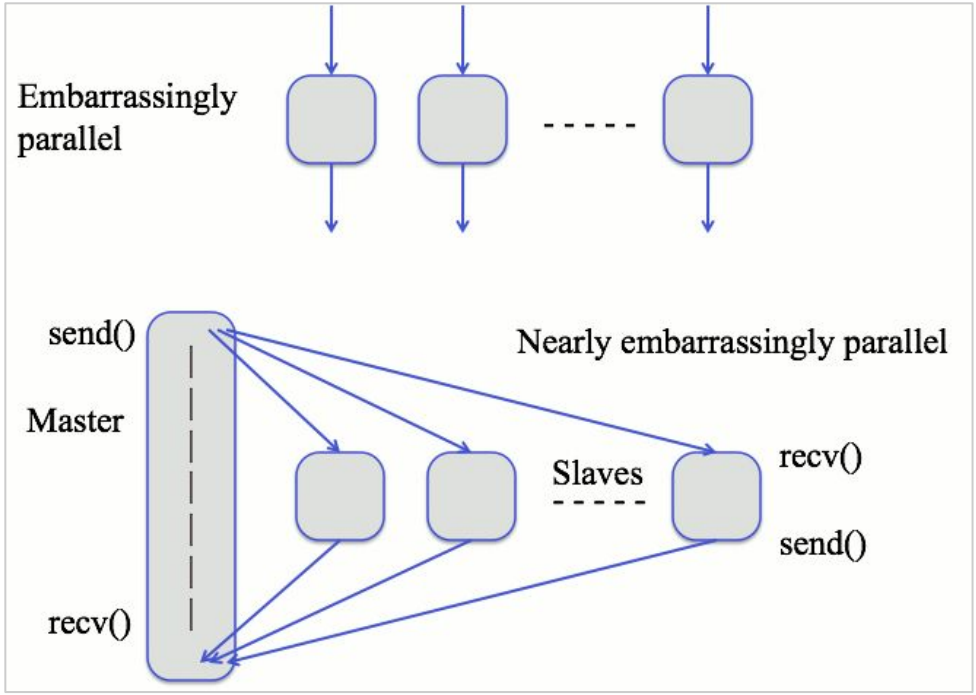


Figure 3.21 Hybrid decomposition for finding the minimum of an array of size 16 using four tasks.

Trivia: Algoritmos vergonzosamente paralelos

- Problema que puede ser dividido en tareas que son completamente independientes
- Las tareas no necesitan comunicarse entre sí
  - O, necesitan muy poca comunicación
- Ej.: SETI@Home (BOINC)



Límites a la paralelización: Las comunicaciones

Estrategias para reducir las comunicaciones:

- Cambiar la formulación del problema
- Cambiar de método numérico
- Duplicar ciertos cálculos
- Modificar el método numérico
- Usar una granularidad más gruesa

Estrategias para acelerar las comunicaciones:

- Comprar una red con más ancho de banda (ej.: 10G Ethernet, Infiniband)
- Comprar una red con menos latencia
- Usar memoria compartida

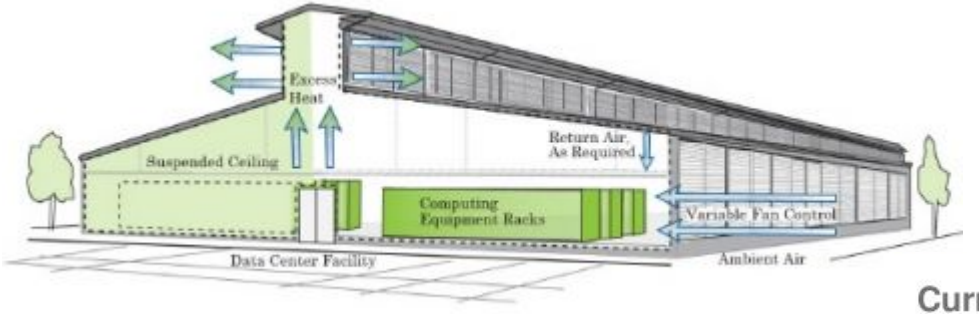
Terminología: Procesamiento masivo

- Uso de gran # de procesadores para la resolución de un problema
  - Hoy en día, más de 1000

- ¿Cómo se implementa?
  - Con un clúster normal, o
  - Puede requerir interconexiones especiales (ej.: Infiniband)
  - Ej.: Blue Gene de IBM

Sidenote: Enfriamiento de datacenters

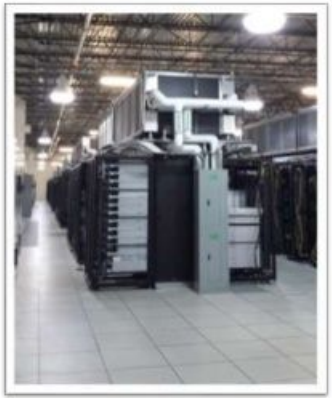
Ejemplo: Diseño de datacenters de Yahoo



Current Notable Designs

Yahoo Computing Coop (YCC)

- 100% Adiabatic Design
- PUE 1.08 – 1.12



Yahoo Thermal Cooling (YTC)

- No Fan Design
- PUE 1.28 - 1.35

PUE: Power Usage Effectiveness

Ejemplo: NCSA's Blue Waters

The Blue Waters system is completely water-cooled. By putting the water almost in contact with the chips that are dissipating the power, it literally conducts the heat directly out without having to move any air around. That allows the electronics to run cooler, which causes them to dissipate less power. Less power dissipated, means less power or energy is used.

During a large part of the year we'll run with what's called free cooling. With the outdoor water-cooling tower, the water won't be any cooler than the outside air temperature but during a large portion of the year the weather is cool. So we'll literally circulate the water through the electronics, outside through a big radiator with a fan blowing through it, and right back into the building.

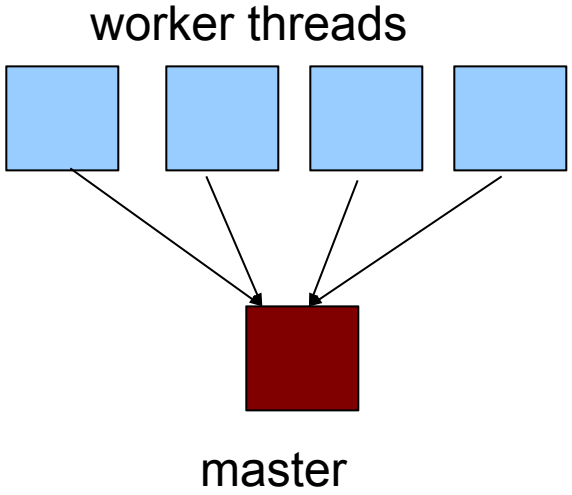
Patrones de paralelismo

Patrones

- Existen algunos patrones que comúnmente se pueden usar para paralelizar tareas:
  - Maestro/Esclavos
  - Productor/Consumidor
  - Colas de trabajo
  - MapReduce

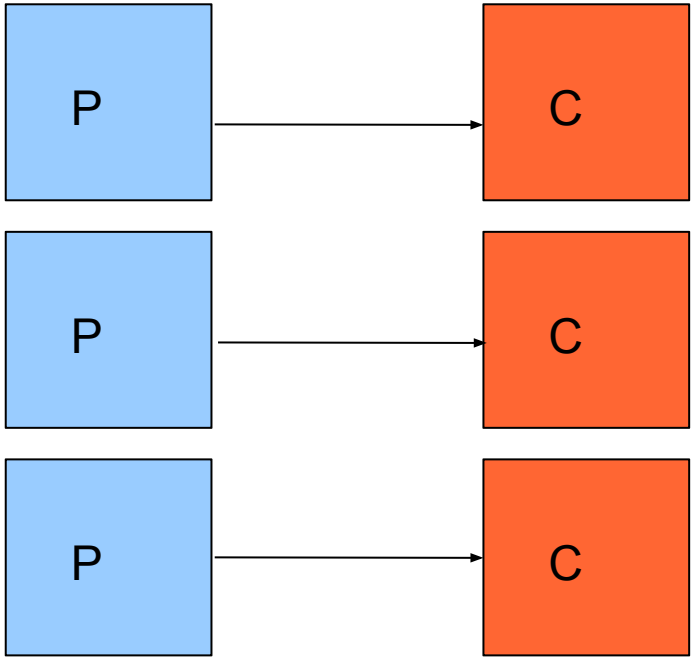
Maestro/esclavo

- Un objeto llamado **maestro** inicialmente es dueño de los datos
- Maestro crea varios **esclavos** (trabajadores) para procesar los elementos individuales
- Maestro espera a que esclavos le reporten los resultados



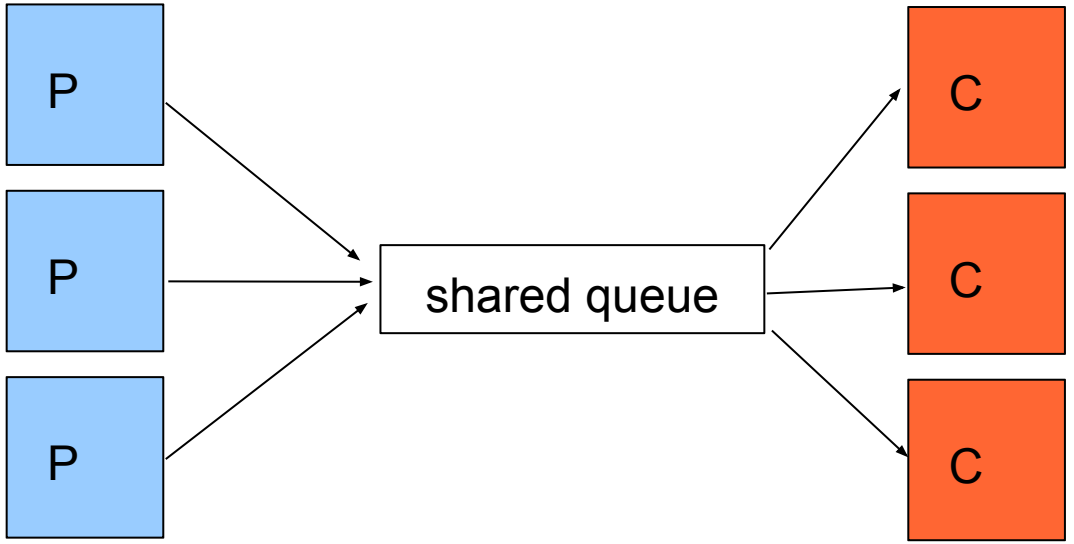
Flujo productor/consumidor

- Productor produce ítems de trabajo
- Consumidor lee dichos ítems
- Puede ser encadenado (**daisy chained**)



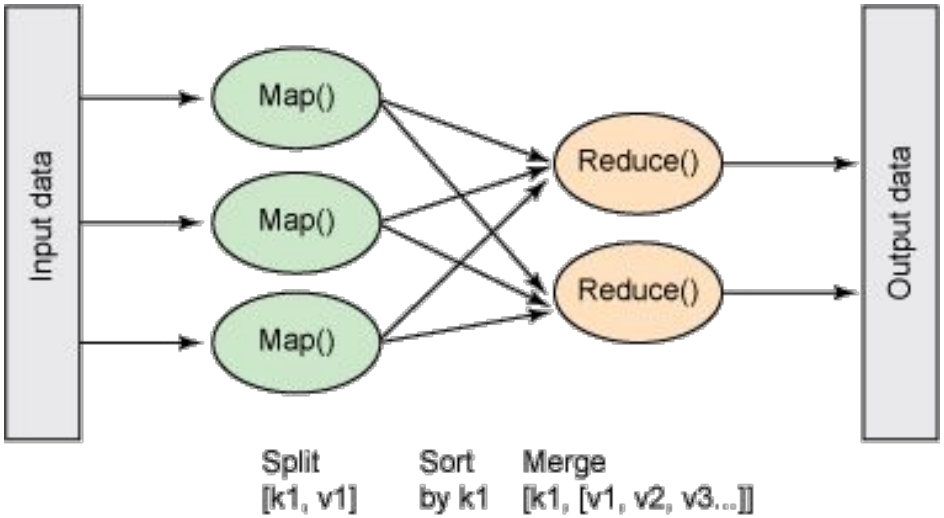
Colas de trabajo

- Múltiples productores y múltiples consumidores
- Un consumidor disponible/libre debe poder procesar cualquier trabajo
- Las colas de trabajo permiten divorciar la relación 1:1 de productores con consumidores
  - “Loosely coupled”



MapReduce

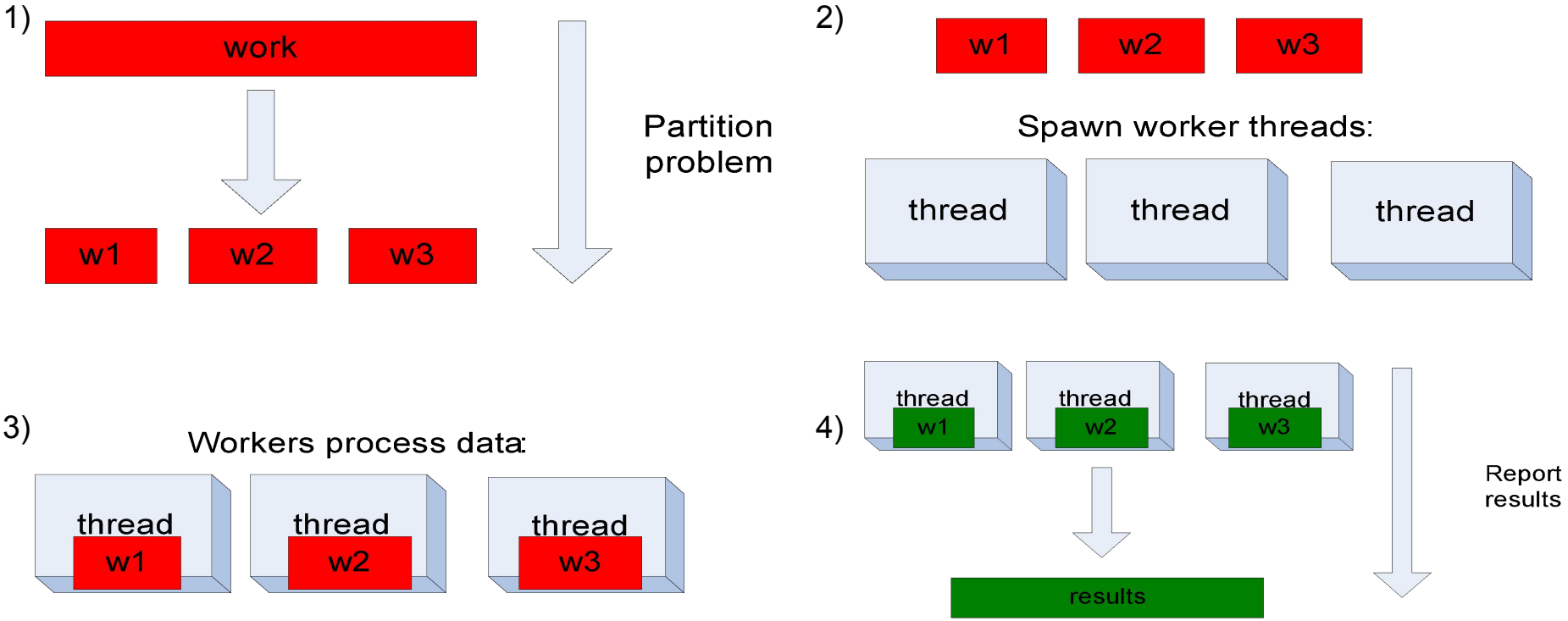
- Incluye el modelo/paradigma y el middleware
  - Modelo/paradigma: MapReduce
  - Middleware: Google’s MapReduce, Apache Hadoop, otros
- Lo estudiamos en detalle, como caso de estudio



Caso de estudio: MapReduce

Preludio: Motivación

Abstracción de paralelización



Problemas con esta abstracción

¡Este modelo es demasiado simple!

- ¿Cómo asignar unidades de trabajo a hilos/procesos?
- ¿Y si tenemos más unidades de trabajos que procesos?
- ¿Cómo agregamos los resultados al final?
- ¿Cómo sabemos que todos los procesos han terminado?
- ¿Y si el trabajo no puede ser dividido en tareas completamente independientes?

Problemas con esta abstracción (cont.)

- Cada uno de estos problemas representa un punto en el que múltiples procesos deben comunicarse unos con otros, o accesar un recurso compartido

- Regla de oro: Cualquier recurso que puede ser usado por múltiples procesos al mismo tiempo necesita de un sistema de sincronización
  - (En realidad no hace falta si los accesos son de solo-lectura)

Solución tradicional: Primitivas de sincronización

- Primitiva de sincronización → variable compartida especial, cuyo acceso se garantiza que es **atómico**
- Implementadas con soporte de HW
- Ejemplos:
  - Semáforos
  - Variables de condición
  - Barreras
- Mucha sincronización → interbloqueos :-(

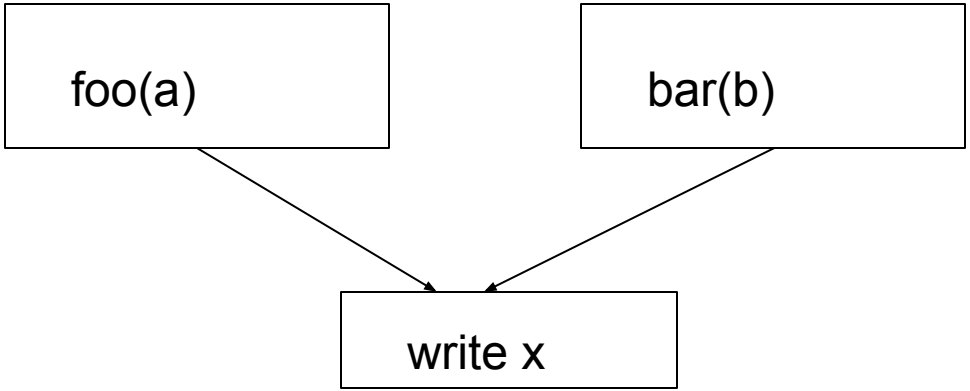
Problema

- **Sincronizar código es difícil**
  - Se debe considerar todas los posibles estados compartidos
  - Se debe mantener organizados los candados y usarlos consistente y correctamente
- Saber que hay bugs no es fácil; corregirlos es aún más difícil
- Minimizar la cantidad de estado compartido permite reducir la complejidad del sistema

Solución

- ¡No compartir nada!
- Ejemplo: MapReduce
- Aplicable a paralelismo de datos
  - Aplicaciones independientes pueden ser conmutadas y ejecutadas en paralelo

```
x :=  foo(a) +  bar(b)
```



MapReduce

- Paradigma diseñado por Google para permitir que un subconjunto de problemas distribuidos sean más fáciles de codificar
  - Dicho subconjunto es bastante grande
  - Big Data
  - Basado en conceptos de la programación funcional
- Automatiza la distribución de los datos y la agregación de los resultados
- Restringe las maneras en que los que se puede interactuar con los datos
  - Si no hay estado compartido → No hacen falta candados (sincronización)

MapReduce

Diapositivas en:

https://research.google.com/archive/mapreduce-osdi04-slides/index.html