Алгоритмы работы распределенных систем

Судаков А.А.

“Параллельные и распределенные вычисления” Лекция 18

План

- Балансировка нагрузки
- Выбор координатора
- Модели консистентности
- Создание общего состояния
- Распределенные блокировки

Балансировка нагрузки

- Дисбаланс уменьшает эффективность вычислений
  - Самый медленный процессор параллельной схемы заканчивает свою работу последним
  - Все остальные в это время простаивают
- Балансировка нагрузки - равномерное распределение нагрузки на процессоры

Характеристики загруженности процессоров

- Чтобы балансировать нагрузку необходимо количественно измерять загруженность процессоров
  - Количество задач на узле (workload)
    - Логична, когда процессоры не перегружены
  - Средняя загруженность процессора (load average)
    - Количество процессов, которые были готовы к выполнению или выполнялись в течение интервала времени (1минута, 5 минут, 15 минут)
    - Чем больше процессов выполняется, тем меньше времени уделяется каждому процессу
  - Эффективная загруженность процессора
    - Отношение средней загруженности процессора к производительности процессора
    - Более быстрый процессор выполняет одну и ту же работу быстрее, чем более медленный

Методы балансировки нагрузки

- Статические
  - Распределение работы между процессорами выполнятся на этапе запуска программ
- Динамические
  - Распределение выполняется в процессе работы программ

Алгоритмы статической балансировки нагрузки

- Круговой алгоритм (round robin)
  - Задачи запускаются по очереди на каждом процессоре, который удовлетворяет необходимым требованиям
- Стохастический алгоритм
  - задачи запускаются на случайно выбранном процессоре
- Распределение данных и функций
  - На этапе запуска определяется оптимальное количество данных и операций, которые будет выполнять каждый процессор процессор
- Распределение на основании целевой функции
  - Задачи запускаются на том узле, для которого целевая функция имеет экстремум

Эффективность статического распределения

- Эффективно при малом времени работы программ
- При большом времени накапливается дисбаланс нагрузки
- Не требует специальной поддержки программ и операционной системы
- Используется в системах пакетного режима для Beowulf кластеров

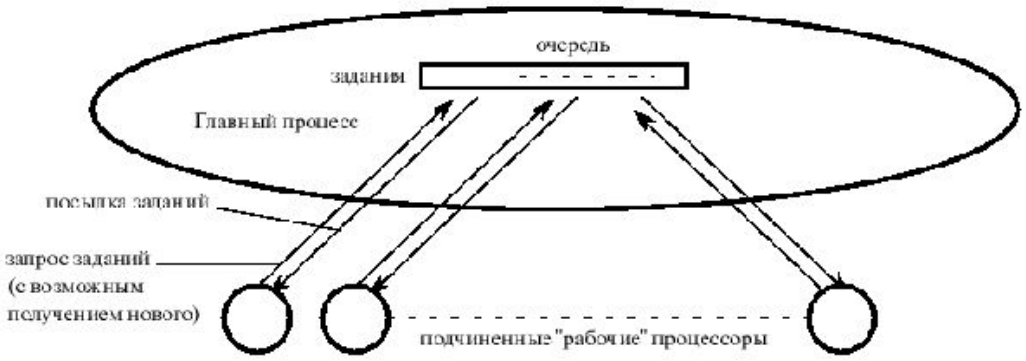
Методы динамической балансировки нагрузки

- Централизованная схема
- Децентрализованная схема
- Балансировка на основе миграции процессов

- Требуют специальных возможностей от программного обеспечения или операционной системы

Централизованная схема

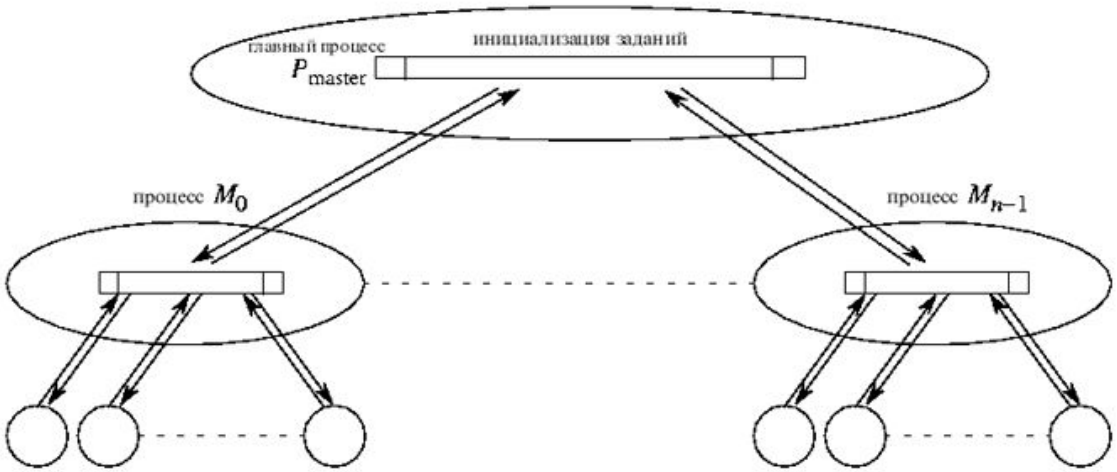
- Один процессор – главный
- Остальные – рабочие
- На главном поддерживается очередь задача
- Задачи раздаются рабочим процессорам по мере выполнения ими предыдущих заданий
- Более быстрый процессор выполнит больше работы
- Требует специальной поддержки программ



Централизованная динамическая балансировка

Децентрализованная схема

- Несколько главных процессоров
- На первом этапе инициализация заданий
- Далее каждый процессор выполняет балансировку по централизованной схеме
- Более эффективна, чем централизованная схема – меньше узких мест



Балансировка на основании миграции процессов

- Процессы во время выполнения мигрируют между процессорами для обеспечения балансировки нагрузки
- Эффективна для задач, которые выполняются длительное время и мало обмениваются между собой

Оптимизация использования ресурсов

- Вводится количественная характеристика использования ресурсов
- Вводится целевая функция, которая принимает большие значения при увеличении использования ресурса
- Процессы перемещаются (запускаются) на тех процессорах, для которых целевая функция наименьшая

Целевая функция

- Должна включать все количественные характеристики использования всех ресурсов
- Характеристики использования ресурсов должны быть безразмерными
- Функция должна возрастать при увеличении использования хотя бы одного из ресурсов
- Если использование ресурса превышает максимально допустимое значение, то функция должна сильно возрастать

Безразмерные характеристики использования ресурсов

- Ресурсы – память, процессор, сеть, диск
- Каждый ресурс имеет свою количественную характеристику
  - Процессор – эффективная загруженность
  - Память – объем свободной памяти
  - Сеть – объем передаваемой информации
- Безразмерность обеспечивается введением отношения текущего значения ресурса к его максимально-допустимому значению
  - Память – объем памяти машины
  - Процессор – максимальная эффективная загрузка

Вид целевой функции

$$f = \sum_{k=0}^{N_{res}} p^{\frac{r_i}{r_i^{\max}}}$$, де

p — кількість процесорів, N_{res} — кількість ресурсів, r_i — значення характеристики i-го ресурсу, $r_i^{\max}$ — максимально допустиме значення характеристики i-го ресурсу.

- F – цена возможного состояния
- Степенная функция – очень быстро растет, когда показатель становится больше единицы
- При увеличении количества процессоров значение функции уменьшается

Результаты измерения производительности при большом количестве невзаимодействующих процессов

	Круговий алгоритм	Розподіл за мінімумом ціни можливості	Динамічне балансування з міграцією процесів
Середнє для всього виконання	29.98788	16.29643	13.67707
Середнє для одного завдання	33.31620	16.76646	14.00990

Выбор координатора

- Во многих случаях возникает задача выбора одной главной машины среди равноправных – координатора (инициатора)
- Это позволяет динамически реализовать централизованные схемы
- Применение
  - Протокол NetBIOS - координатор выбирается для службы имен
  - Сеть Token Ring – одна машина выбирается координатором для обмена маркером

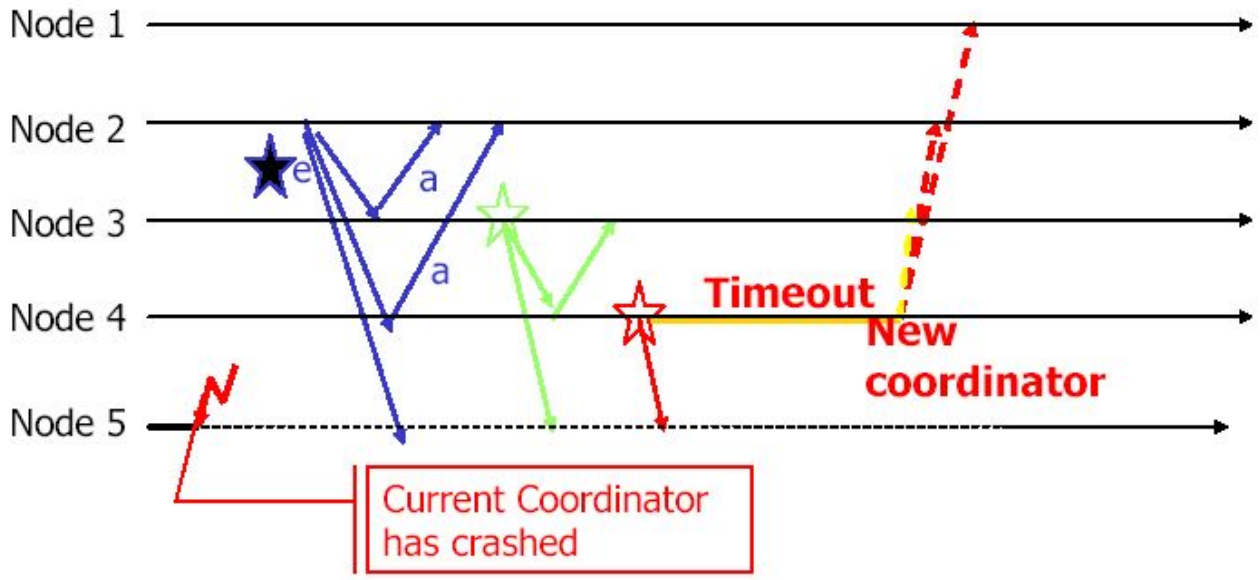
Условия

- Есть произвольное количество процессов, которые могут взаимодействовать между собой
- Каждый процесс имеет уникальный номер
- Выбрать процесс с наибольшим номером и сообщить номер этого процесса всем
- В случае выхода из строя координатора алгоритм должен выбрать нового координатора

Алгоритм задиры (bully algorithm) 1982

- Процесс `i` обнаружил пропажу координатора и отправляет сообщение `E` о начале выборов всем узлам (с большим номером)
- Если процесс `i` не получил ни от кого ответа `A` в течение интервала времени `T`, то он назначает себя координатором и отправляет всем узлам (с меньшим номером) сообщение `C` – выбран новый координатор
- Если процесс `j` получил сообщение о начале выборов `E` (от процесса с меньшим номером) то но отвечает ему сообщением `A` и запускает алгоритм для себя
- Если процесс i получил ответ `A` от узла `j`, то значит, что узел `j` имеет больший номер и может стать координатором. Если в течение интервала времени `T` не было получено сообщения `C`, алгоритм повторяется через интервал времени `T1`.
- При появлении нового процесса он запускает алгоритм для себя
- Процесс с максимальным номером становится координатором

Пример

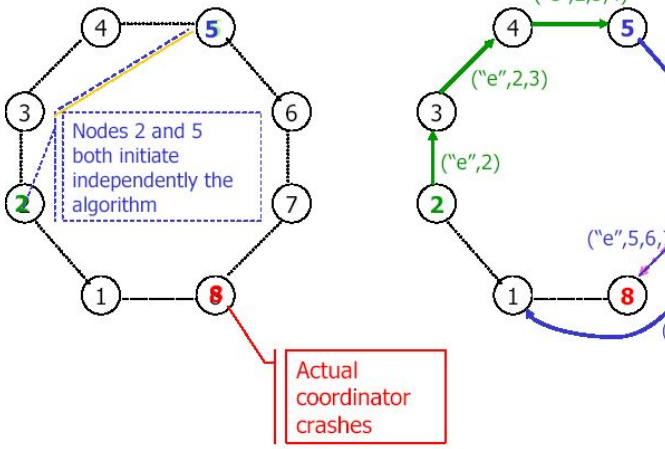


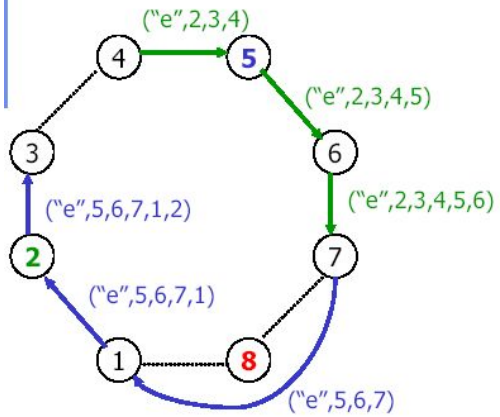
- Узел 2 обнаружил потерю координатора
  - Отправляет сообщение узлам 3,4,5
- Узлы 3 и 4 приняли сообщение узла 2, сказали ему остановиться и запускают алгоритм для себя
- В конце концов узел с максимальным номером запустит алгоритм для себя, назначит себя координатором и расскажет всем

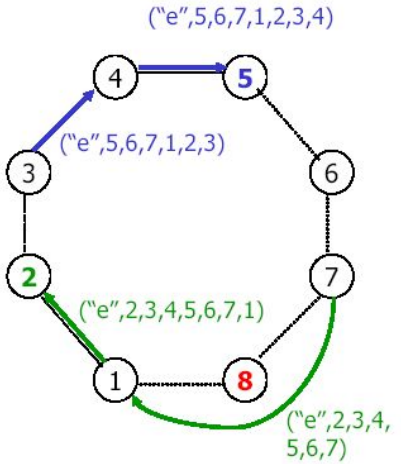
Кольцевой алгоритм (1977)

- Узел `i` обнаружил потерю координатора, он отправляет сообщение `E` со своим номером узлу с номером `i+1` и если нет подтверждения, то узлу с номером `i+2` и т.д.
- Если узел `j` не встретил в `E` сообщении своего номера, то он добавляет туда свой номер и отправляет его дальше
- Если узел `j` встретил в `E` сообщении свой номер, то значит `E` сообщение обошло кольцо и максимальный номер в нем – номер координатора. Узел `j` меняет тип сообщения на `C` и отправляет его дальше
- После того, как `C` сообщение обошло все узлы, оно уничтожается и все узлы знают номер координатора

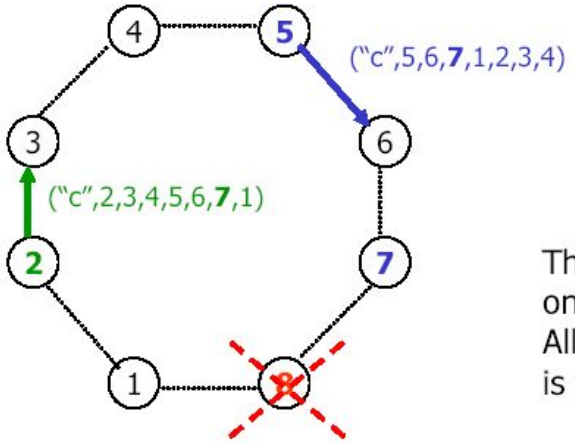
Пример кольцевого алгоритма







Both e-messages circled once around the ring of all active nodes



This coordinator-message circles once around the logical-ring, All nodes know that **7** is the new coordinator

Количество операций

Algorithm	Number of Messages	Time
Bully	$O(n^2)$	O(n)
Ring	2(n-1)	2(n-1)

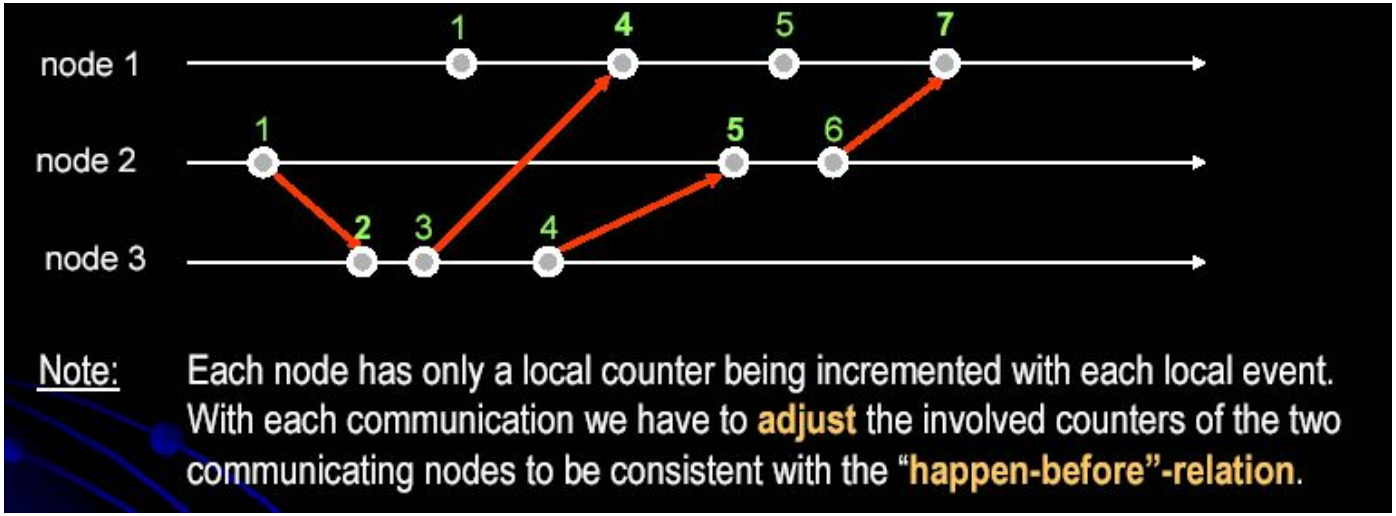
Синхронизация времени

- Лампортовские метки – определение раньше-позже для определенной последовательности событий
- Достаточно для многих проблем

Метки Лампорта

- Поддерживается счетчик событий
- В начальном состоянии от равен 1
- После каждого события счетчик увеличивается на 1
- Если одно событие произошло раньше другого, то Лампортовская метка более раннего события будет меньше

Пример


node 1
node 2
node 3
1
4
5
7
1
5
6
2
3
4
Note:
Each node has only a local counter being incremented with each local event.
With each communication we have to adjust the involved counters of the two communicating nodes to be consistent with the "happen-before"-relation.

Синхронизация в распределенных системах

- Доступ к общим ресурсам в сети требует синхронизации так же как и в случае общей памяти
  - Общий диск
  - Общая распределенная память
- Проблемы
  - Нет действительно общих ресурсов, они только виртуальные
  - Нет общего состояния
  - Нет общего времени
  - Другой тип параллелизма
  - Алгоритмы должны работать на каждом узле (процессе)
  - Возможность выхода из строя частей системы

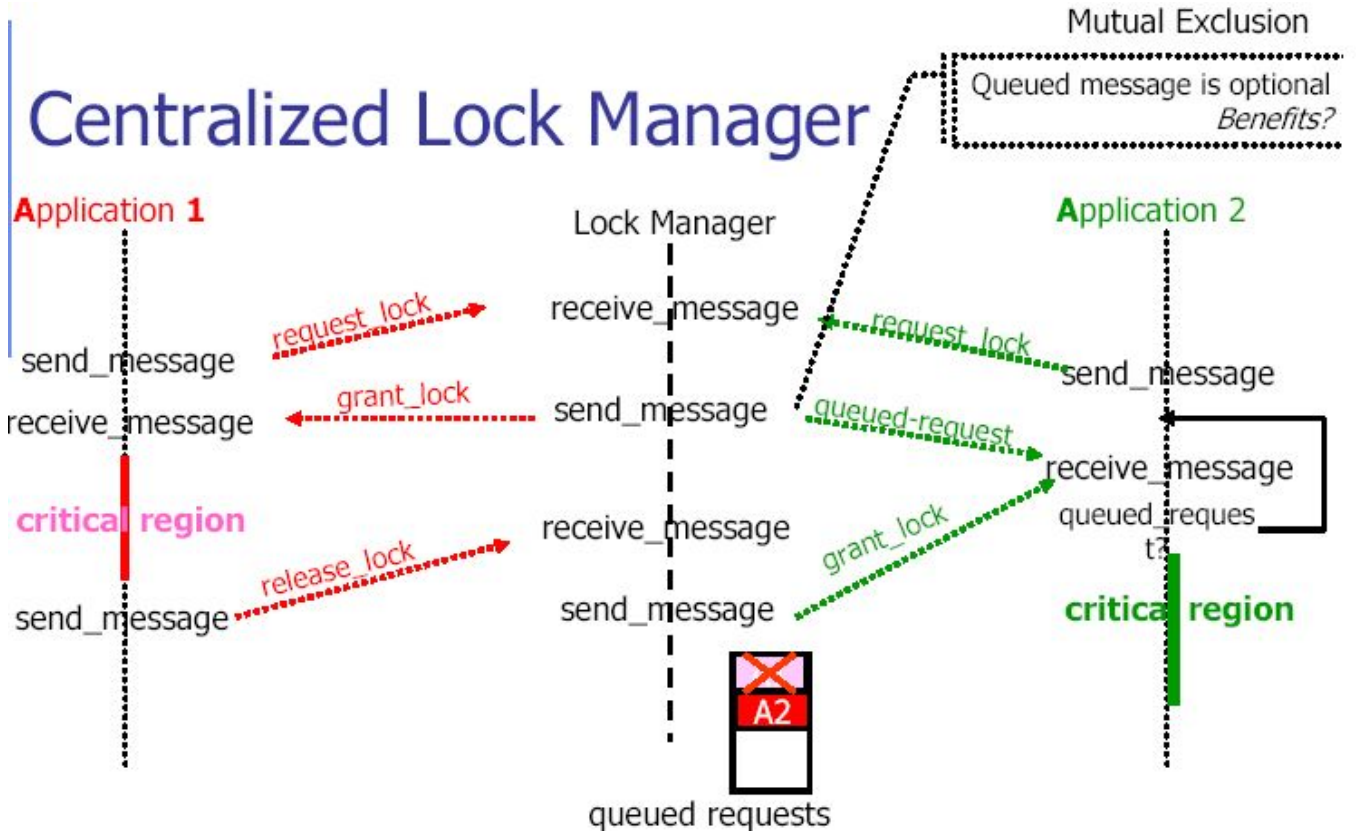
Взаимоисключающий доступ

- По аналогии с общей памятью
  - Есть некоторый общий ресурс (блок файловой системы)
  - Его может изменять только один процесс
  - Необходимо обеспечить блокировку, которая бы давала возможность защитить доступ к ресурсу
- Типы алгоритмов блокировки
  - Централизованный (CLM)
  - С передачей маркера (TLM)
  - Распределенный менеджер блокировок (DLM)

Централизованный алгоритм

- Один из процессов выбирается координатором
- Все доступы к общему ресурсу выполняются через координатора
- Для доступа к ресурсу процессы посылают запрос координатору
- Если ресурс свободен координатор отправляет сообщение о возможности доступа
- Если ресурс занят запрос ставится в очередь
- При освобождении ресурса отправляется запрос о возможности доступа первому в очереди

Пример работы централизованного менеджера блокировок

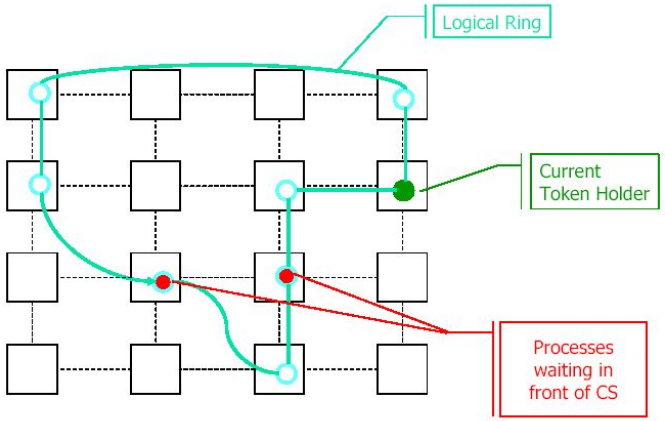


Особенности CLM

- Преимущества
  - Просто реализовать
- Недостатки
  - Единая точка сбоя
  - Плохо масштабируется
  - Возможность возникновения узкого места в плане производительности

Алгоритм Token Ring

- Создается логическое кольцо процессов
- По кольцу передается маркер
- Блокировку захватывает тот, кто получает маркер
- При освобождении блокировки маркер передается дальше



Особенности

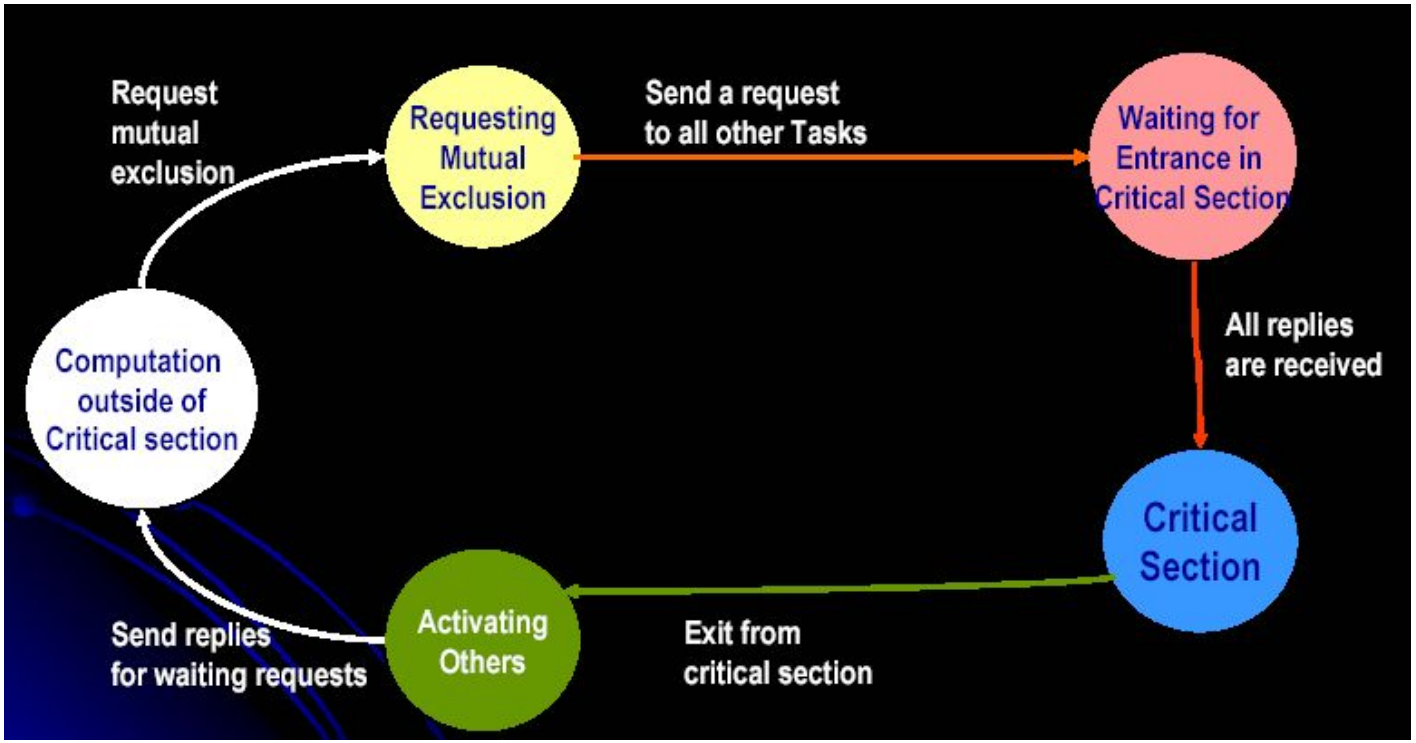
- Преимущества
  - Простой и обеспечивает взаимоисключающий доступ
- Недостатки
  - Ненадежный
    - Если маркер недоступен длительное время, то его удерживают, или он потерян?
    - Как добавить новых участников?
  - Не поддерживается порядок захвата блокировки
  - При большом количестве узлов получаются большие задержки

Распределенный менеджер блокировок

- Каждый узел может обмениваться с остальными
- Возможны три типа сообщений
  - REQUEST
  - QUEUED
  - GRANT
- Центральным узлом для блокировки является узел, захвативший блокировку
- Узлы могут быть в трех состояниях
  - Удержание блокировки
  - Ожидания
  - Свободное состояние

Алгоритм работы

- Каждый узел выполняет общий алгоритм



Распределенный захват блокировки

- Алгоритм захвата блокировки
  - Узел `i`, который желает захватить блокировку отправляет сообщение `REQUEST` всем остальным `n-1` узлам (multicast)
  - Все узлы отвечают `i`-му `n-1` сообщением, которые могут быть следующего типа
    - `GRANT` – разрешение захвата - ответ узла, который не удерживает блокировку
    - `QUEUED` - информация, что запрос поставлен в очередь на узле, который захватил блокировку
  - После получения `n-1` сообщения узел `i` выполняет следующие действия
    - Если все сообщения были типа `GRANT`, то узел `i` считается владельцем блокировки
    - Если одно сообщение типа `QUEUED`, то узел ждет сообщения типа GRANT

Конфликт при захвате

- Если после отправки сообщения `REQUEST` узел получает сообщение `REQUEST` с узла `j` (или других узлов) до того, как получены все сообщения от других узлов
  - Каждое сообщение содержит Лампортовскую метку времени
  - Если метка полученного сообщения меньше, чем локальная, то узлу `j` отправляется сообщение GRANTED
  - Иначе запрос ставится в очередь и отправляется сообщение QUEUED

Распределенное удержание блокировки

- Алгоритм работы узла в состоянии удержания блокировки
  - Узел j захватил блокировку и поддерживает очередь запросов на захват
  - Если получено сообщение `REQUEST` от `i`-го узла, то
    - Установить в очередь запрос от узла `i`
    - Отправить узлу i сообщение `QUEUED`

Освобождение распределенной блокировки

- Алгоритм освобождения блокировки
  - Если в очереди `i`-го узла есть запросы на захват, он отправляет сообщение `GRANTED` первому в очереди и удаляет его из очереди

Восстановление после сбоя узла

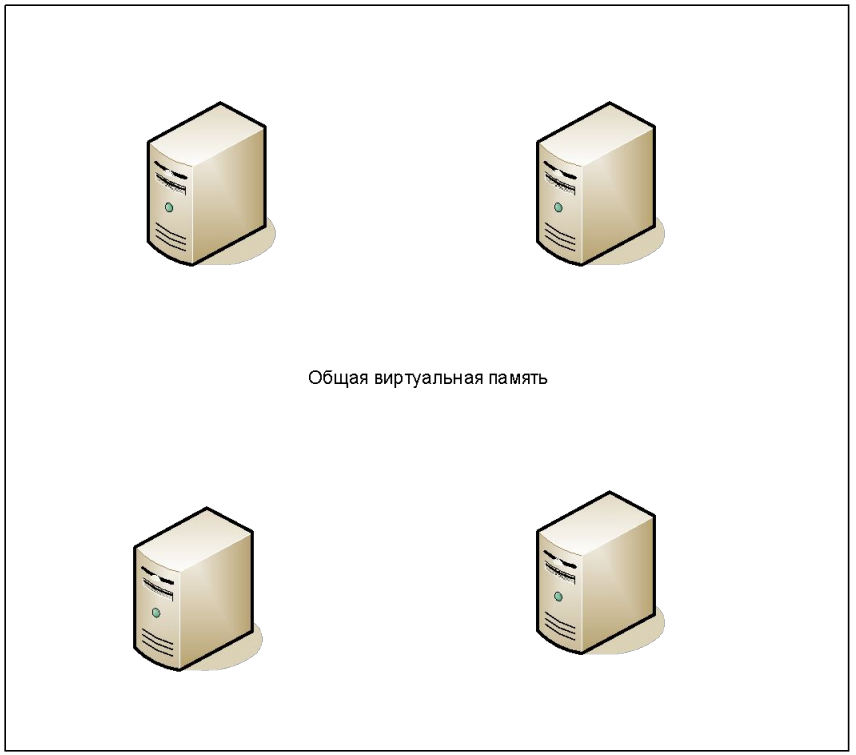
- Вышел из строя узел, который не удерживает блокировку
  - Количество участников уменьшается на 1
- Вышел из строя узел, который удерживает блокировку
  - Этот узел отключается от операций с общим ресурсом (fence)
  - Восстанавливается состояние общего ресурса из журнала
  - Количество участников уменьшается на 1
  - Все очереди уничтожаются
  - Запускается новый процесс захвата блокировки

Сравнение алгоритмов блокировки

Алгоритм	Сообщений на один критический участок	Задержка	Потенциальные проблемы
Централизованный	3	$2t_o$	Выход узла из строя, одна точка сбоя
Кольцевой	2n-1	$(2n-1)t_o$	потеря маркера, выход из строя узла
Голосование	2n-2	$(2n-2)t_o$	выход из строя узла

Консистентность

- Консистентность – поддерживание общих ресурсов в распределенной системе в непротиворечивом состоянии
- Пример
  - Эмуляция общей памяти на распределенной системе
  - Работа с общими дисковыми ресурсами
  - Все изменения, сделанные одним узлом не должны привести к некорректном состоянию данных



Модели консистентности

- Строгая (последовательная) консистентность
- Процессорная консистентность (ослабленная консистентность)
- Слабая консистентность
- Косистентность захвата-освобождения

Процессорная консистентность

- Все операции чтения записи одного процессора должны выполняться строго последовательно, но порядок операций разных процессоров может меняться
- Реализуется путем передачи данных пакетами (блоками)

Строгая (последовательная) консистентность

- Все операции чтения-записи общего ресурса должны выполняться в строго в той последовательности, в которой поступил запрос (как в случае машины с общей памятью)
- Реализуется по централизованной схеме
- Не эффективна

Слабая консистентность

- Доступ разделяется на операции чтения-записи и операции синхронизации. Все операции синхронизации выполняются последовательно по отношении друг к другу
- Данные отправляются пакетами, каждый из которых консистентный для данного процессора

Косистентность захвата-освобождения

- Доступ к ресурсу разделяется на операции захвата и освобождения
- После захвата (GET) можно эксклюзивно обращаться к ресурсу
- После освобождения (PUT) ресурс может захватить другой процесс
- Два подхода
  - Агрессивный подход – PUT синхронизирует данные
  - Ленивый подход - GET синхронизирует данные

Другие модели консистентности

- Консистентность областей
  - Ресурс разбивается на области и для каждой области используется своя модель консистентности
- Консистентность структур данных
  - С каждой структурой данных связывается своя модель консистентности

Алгоритмы сохранения общего консистентного состояния (snapshot)

- Общее состояние
  - запись данных распределенной файловой системы
  - Checkpoint распределенной программы

Алгоритм Ченди-Лампорта

- Любой процесс распределенной системы может инициировать создание слепка
- Инициатор передает сообщение всем участникам о начале записи общего состояния
- При получении сообщения начинается запись общего состояния
- После записи общего состояния начинается запись всех сообщений от всех процессов, пока не будет получено сообщение о завершении записи состояния
- Продолжить работу

Вопросы?