Операционные системы

Курс лекций для гр. 4057/2

Лекция №11

Вопросы к лекции 10

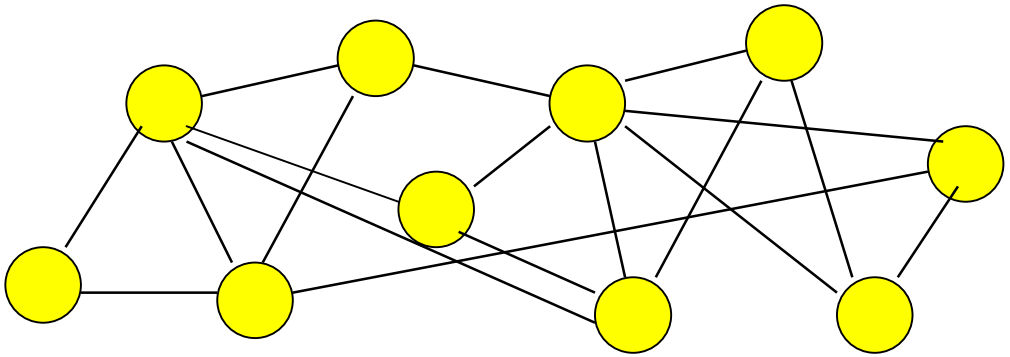
1. Сравните эти два подхода (хранение дескрипторов файлов в каталогах и в индексном файле) с точки зрения удобства, эффективности и надежности.
2. Некоторые ОС автоматически открывают файл при первой ссылке на него и закрывают при завершении процесса, если не осталось других процессов, его использующих. Каковы преимущества и недостатки этой схемы по сравнению с более распространенной, когда пользователь должен явно открывать и закрывать файл?
3. Почему для блочного кэша диска оказалось возможным реализовать точный LRU, а в виртуальной памяти – нет?
4. Что следует указать диспетчеру кэша: в конец или в начало списка простаивающих страниц (т.е., FIFO очереди отложенного вытеснения) следует помещать условно вытесненную страницу в случае последовательного доступа к файлу? Прямого доступа?

Содержание

Основные понятия

Распределенная система — это множество соединенных сетью связи компьютеров, на которых выполняются взаимосвязанные процессы. Общей памяти у них нет.

Узел сети – компьютер или выч. система



Коммуникация в распределенных системах - с помощью передачи сообщений в локальной или глобальной сети

Цели распределенных систем

- Предоставление вычислительного сервиса на расстоянии
- Повышение производительности
  - благодаря распараллеливанию вычислений
- Повышение надежности системы
  - благодаря резервированию
- Лучшая масштабируемость системы

Примеры

- Банк со многими филиалами
- Система бронирования билетов
- Роботизированный цех
- Система управления транспортным средством
- WWW
- Глобальная поисковая система
- Вычислительный кластер

Уровни программной поддержки распределенных вычислений

1. Сетевая архитектура
   - Поддержка цифровой связи: e-mail, ftp, удаленные терминалы
   - Машины автономны и имеют собственную ОС
   - Поддержка: комплект протоколов TCP/IP
2. Сетевая ОС
   - Конфигурация: сеть клиентских машин + сервер(ы)
   - Сетевая ОС – надстройка над локальной ОС каждой машины
   - Поддержка: сетевой сервис RPC, распределенная файловая система
3. Распределенная ОС
   - Сеть машин совместно использует общую ОС
   - С точки зрения пользователя эта сеть выглядит как единый компьютер
   - Доступ к ресурсам системы прозрачен для пользователей

Два основных вида распределенных систем

1. Узлы сети территориально удалены друг от друга. Цели системы:
   - предоставление вычислительного сервиса на расстоянии
     - модель *несимметричного* взаимодействия «клиент/сервер»
   - обмен данными
     - обмен через посредника (напр., e-mail) – модель «клиент/сервер»
     - прямая связь равноправных узлов – пиринговая сеть (peer-to-peer)
   - Пример: корпоративные информационные системы (ERP)

2. Узлы - процессоры расположены в одном помещении
   - Главная цель – повышение производительности
   - Пример: вычислительный кластер
   - В отличие от многопроцессорных и многоядерных систем, у этих процессоров нет общей памяти !
   - Большинство процессоров равноправны и выполняют части общей задачи (трудоемкого численного метода или мощного сервера)
   - В суперкомпьютере процессоры связаны быстрыми шинами, а в кластере – быстрой локальной сетью

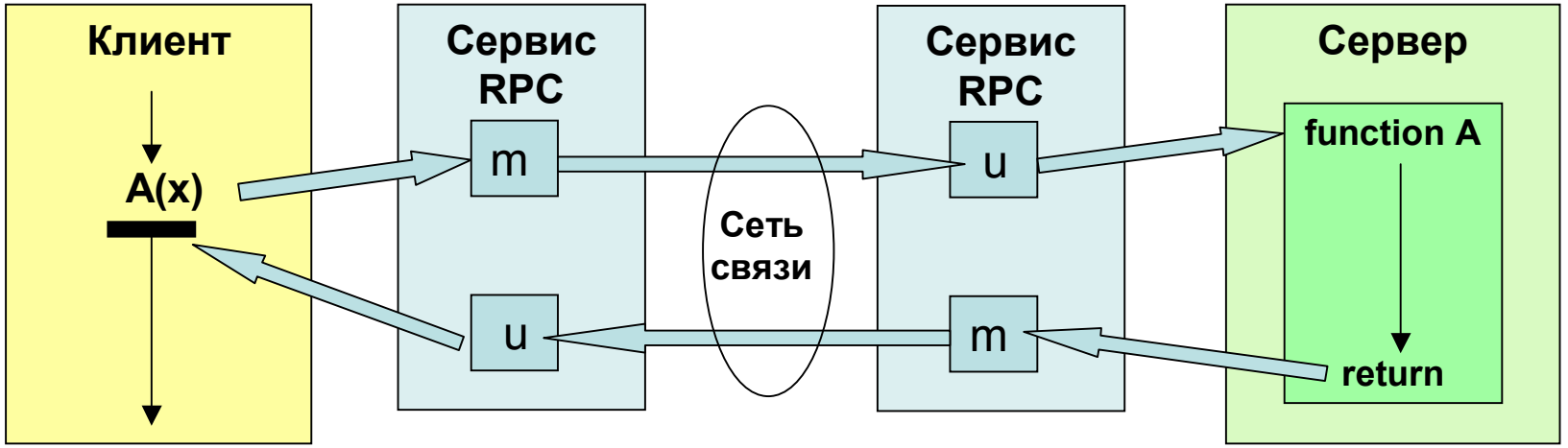
     NB: существуют суперкомпьютеры, где процессоры имеют общую память – тогда они относятся к сосредоточенным, а не распределенным системам

Модель клиент-сервер

Сервер – процесс или набор процессов на машине, хранящей данные, с которыми работают клиенты, предоставляющий некоторый вычислительный сервис (работу приложений)

Клиент – программа, использующая сервис

Общепринятый метод коммуникации – удаленный вызов процедур (RPC):



- Клиент сначала связывается с сервером, т.е. находит его в сети и устанавливает соединение
- Затем клиент посылает серверу запрос выполнить действие, синтаксически идентичный локальному вызову процедуры, а сервер выполняет процедуру в отдельном потоке и возвращает результат

Шаги работы RPC

1. Клиент вызывает локальную DLL, содержащую *заглушку (stub)*, которая имеет то же имя и тот же интерфейс, что и удаленная
2. Заглушка преобразует переданные ей параметры для передачи по сети - упорядочивает и упаковывает их; такой процесс называется *маршалингом (marshaling)*
3. Аргументами процедуры являются объекты данных в стеке и куче вызывающего процесса (на практике все параметры всегда передаются по значению)
4. Заглушка вызывает процедуры библиотеки RPC периода исполнения, они находят сервер, на котором расположена вызываемая удаленная процедура, и посылают запрос
   - Динамический поиск нужного порта в сети осуществляется с помощью *сервиса именования* в данной сети; поиска можно избежать, связав клиента с сервером статически
5. Сервер выполняет обратное преобразование параметров *(unmarshaling)* и вызывает процедуру
6. После ее завершения сервер выполняет обратную последовательность действий для возврата результатов вызывающей программе

Развитие RPC для ООП

Windows

COM/DCOM (Component Object Method)

- вместо процедур вызываются методы объектов
- имена объектов являются глобальными
- в сети есть сервис именования

.NET

- позволяет быстро создавать программируемые веб-сервисы – приложения с веб-браузерным UI для клиента

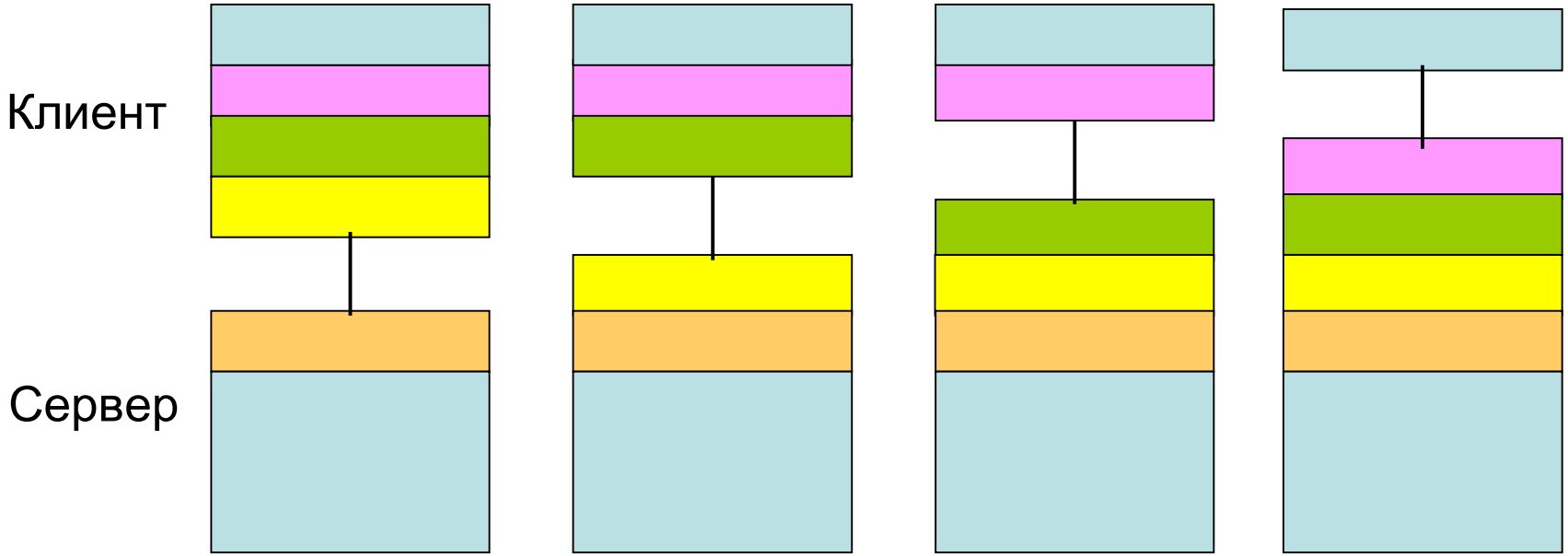
Java

RMI (Remote Method Invocation) – удаленный вызов методов

- информацию о местонахождении и интерфейсе удаленного объекта клиенты получают из реестра RMI (это тоже удаленный серверный объект)
- каждый сервер помещает в реестр сведения о своих объектах, доступных для удаленного вызова

Варианты распределения задач между клиентом и сервером

Логика представления - интерфейс пользователя
Логика приложения (например, бизнес-логика подсистемы корпоративной АСУ)
Логика данных – подсхема модели данных, используемая клиентом в данный момент
База данных + СУБД



Тенденции

Grid-сети

Большое число обычных компьютеров объединяются сети с помощью Интернета для решения трудоемких научных задач в фоновом режиме

- Большие задачи разбиваются на слабо связанные подзадачи
- Планируется их выполнение на неоднородных компьютерах с плохо предсказуемой загрузкой
- Сети конфигурируются динамически

Примеры проектов: SETI (поиск внеземных цивилизаций), расшифровка генома

Облачные вычисления (Cloud computing)

- Среда связи – Интернет
- Серверы – держатели Веб-сервисов
- Клиент - графический терминал + веб-браузер
- Программы арендуются на время, а не покупаются навсегда

Понятие вычислительного кластера

Кластер (cluster=гроздь) - это группа совместно работающих компьютеров – узлов (обычно PC), представляющая собой единый ресурс, единую вычислительную систему

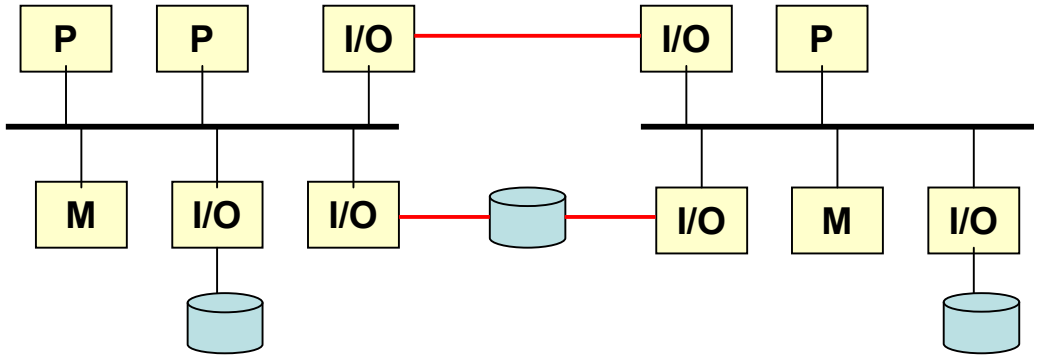


Схема кластера из двух узлов

Сравнение с суперкомпьютерами:

- достоинства: хорошая масштабируемость и дешевизна
- недостатки: большие размеры и энергопотребление

Программное обеспечение кластеров

Расширения ОС

- MS Windows Compute Cluster Server
  - поддерживает до 32 узлов
  - нет совместного использования ресурсов
- Beowulf (Unix/Linux)
  - до нескольких сот машин
  - возможно совместное использование дисков

Стандарт коммуникационного ПО: MPI (Message passing interface)

- API для описания взаимодействия параллельных процессов - ветвей параллельной программы
- Не зависит от машинной архитектуры (однопроцессорные/ многопроцессорные, с общей/раздельной памятью)
- Реализации в виде библиотек: MPICH, LAM, HPVM - для Win32, Unix/Linux

Принципы MPI

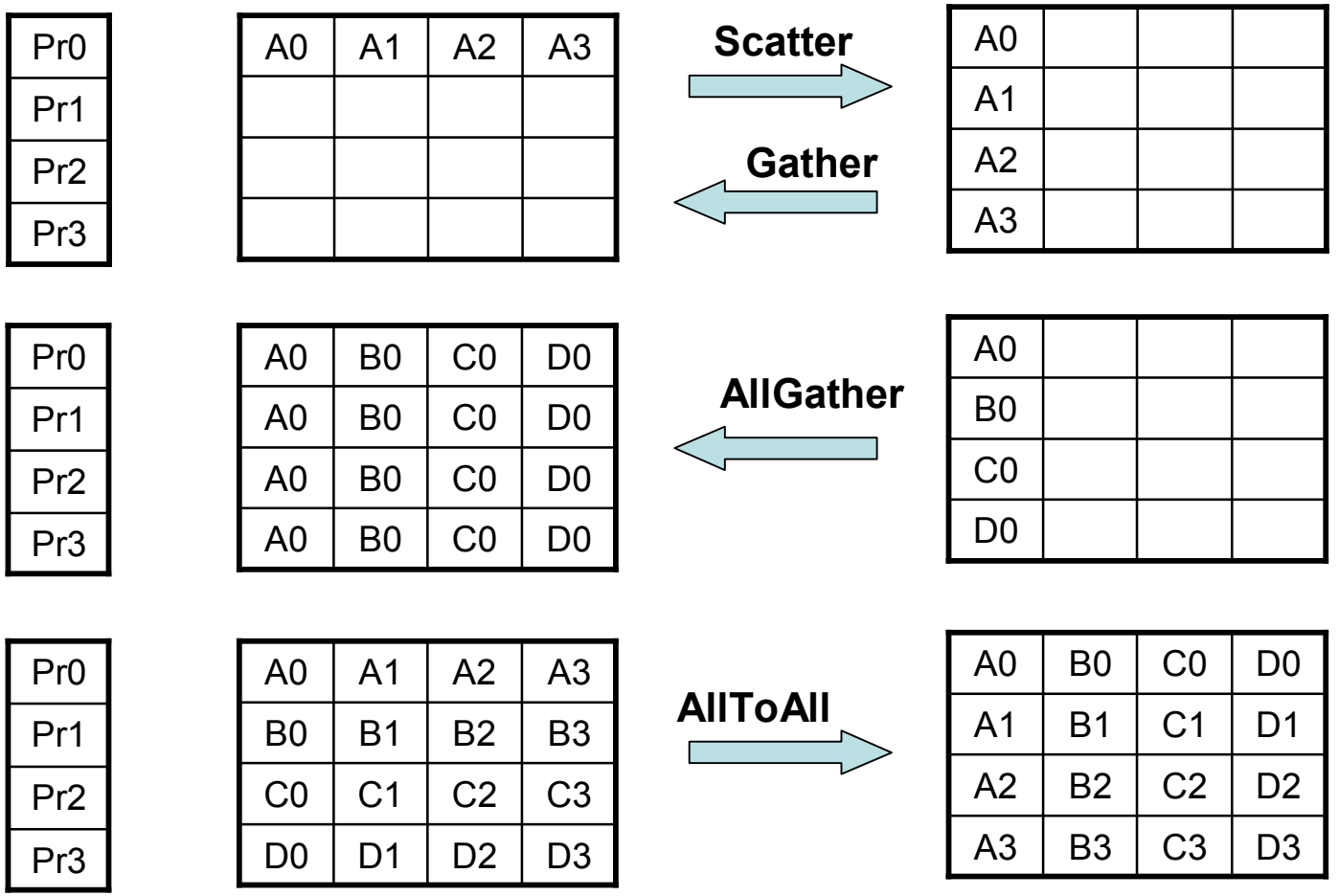
- Для MPI пишется программа, содержащая код всех ветвей сразу
- Если MPI-приложение запускается в сети, запускаемый файл приложения должен быть построен на каждой машине
- MPI-загрузчиком запускается фиксированное количество процессов - экземпляров программы, обычно равное числу узлов
- Каждый экземпляр определяет свой порядковый номер - *ранг*, и в зависимости от него выполняет ту или иную ветвь алгоритма
- Каждая ветвь имеет адресное пространство данных, полностью изолированное от других ветвей; они обмениваются данными только в виде сообщений
- Число ветвей фиксировано, в ходе работы порождение новых ветвей невозможно

MPI дает простой способ создания процессов для модели SPMD: одна программа используется для обработки разных данных на разных процессорах

Операции MPI

- Группирование процессов: группы служат для указания адресата сообщений (процесс-адресат специфицируется своим номером в группе), определяют исполнителей коллективных операций
- Коммуникация: send, receive:
  - блокирующие и не блокирующие операции
  - с выборочной и широковещательной адресацией
  - с буферизацией или без
  - с фильтрацией по типу сообщения и пр.
- Барьерная синхронизация: функция Barrier – ожидать прихода в точку синхронизации («к барьеру»). Возврат из нее происходит, когда все процессы группы вызовут эту функцию (Вопрос 1)
- Глобальные операции (сумма, максимум, и т.п.): результат сообщается всем членам группы или только одному. Пользователь может сам определить глобальную операцию – функцию
- Функции распределения данных, напр. Scatter – распределить массив a, состоящий из size элементов, отправляя каждому i процессу в группе сообщение со значением a[i]

Функции распределения данных



Пример программы

```
#include <mpi.h>
main(int argc, char *argv[]) {
  int myid, otherid;              //идентификаторы (ранги) процессов
  int size;                        //число процессов
  int length = 1;                 //длина сообщения – число элементов
  int tag = 1;                    //метка типа сообщений
  int myvalue, othervalue;        //обмениваемые значения
  MPI_Status status;              //состояние возврата из функций MPI
  MPI_Init(&argc, &argv);                  //инициализация MPI
  MPI_Comm_size(MPI_COMM_WORLD, &size); //определить число процессов
  MPI_Comm_rank(MPI_COMM_WORLD, &myid);  //определить ранг процесса
  if (myid == 0) {
    otherid = 1; myvalue = 14; }
  else {
    otherid = 0; myvalue = 25; }
  MPI_Send(&myvalue, 1, MPI_INT, otherid, tag, MPI_COMM_WORLD);
  MPI_Recv(&othervalue, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG,
           MPI_COMM_WORLD, &status);
  printf("процесс номер %d получил %d\n", myid, othervalue);
   MPI_Finalize();}
```

Параллельная Виртуальная Машина

PVM – стандарт для распределенных вычислений, предшественник MPI

- Единица параллельной работы в PVM – задача=процесс Unix
- Сначала запускается главная задача (master), которая производит некоторые подготовительные действия, например инициализацию начальных условий
- Далее она запускает остальные задачи (slaves), которым может соответствовать либо тот же исполняемый файл, либо разные исполняемые файлы – в этом отличие от MPI
- Такой вариант организации параллельных вычислений удобнее, если алгоритмы разных задач существенно различаются
- Межзадачная коммуникация беднее, чем в MPI: send/receive только синхронные, нет scatter/gather (однако есть сигналы Unix)

Достоинства по сравнению с MPI - простота, унаследованный от UNIX аппарат процессов и сигналов

Недостатки - низкая производительность и функциональная ограниченность

Проблемы синхронизации параллельных процессов в распределенных системах

Отличие от сосредоточенной системы:

- У узлов нет общей памяти
- В каждом узле – свое время (свои часы - тактовый генератор)
- Связь между узлами не мгновенная, а с существенной задержкой на передачу сообщений, причем в глобальной сети:
  - время передачи соизмеримо с интервалом между важными событиями
  - разброс времени передачи велик, поэтому порядок приема сообщений может не совпадать с порядком их отправления
  - связь ненадежна, сообщения могут быть потеряны
- Любой узел может в любой момент быть выключен или отказать

Усложняется решение задач:

✓ взаимного исключения доступа к глобальным ресурсам

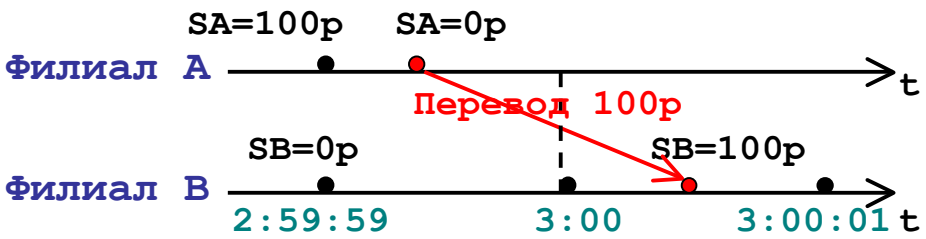
✓ борьбы с взаимоблокировками

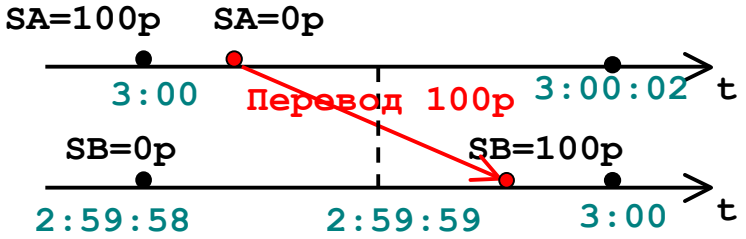
✓ получения непротиворечивого глобального состояния системы

и т. д.

Определение глобального состояния системы

Задача: подсчитать сумму денег на счетах г. N в двух филиалах банка: SA + SB, сегодня в 3:00





Сумма = 0 р

Для избежания этой ошибки в понятие состояния должны входить все сообщения, находящиеся в пути в момент измерения состояния

Сумма = 200р

Здесь часы в филиале В отстают от часов в филиале А на 2 с

Корректное решение этой задачи – атомарные транзакции

Существуют алгоритмы распределенного снимка (snapshot) для получения согласованного глобального состояния

Два вида алгоритмов синхронизации

- Централизованные (сосредоточенные): выполняются на центральном управляющем узле сети, который постоянно собирает информацию о состоянии всех других узлов
  - Центральный узел можно назначать динамически – путем *выборов* узла-координатора

Недостаток: центральный узел – узкое место в смысле надежности и производительности

- Распределенные: выделенного узла нет, алгоритм выполняется на всех узлах параллельно
  - В частности, алгоритм выбора координатора

Отличия распределенных алгоритмов от централизованных

- Ни один из узлов не обладает полной информацией о состоянии системы
- Узлы принимают решения на основе локальной, неполной информации, затрачивая примерно одинаковые усилия
- Отказ одного или нескольких узлов не должен вызывать нарушение алгоритма
- Не требуется предположения о существовании единого времени

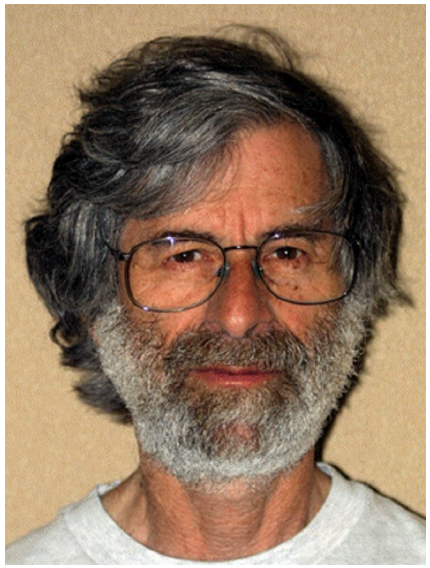
Физическое (абсолютное) время vs. логическое (относительное) время

- Внутренние часы компьютера (кварц) – относительная точность 10^{-6} - уход часов (погрешность): 1 мс за 17 мин
- GPS: точность UCT (Universal Coordinated Time) - 1 мс
- Протокол сетевого времени в Интернете - Network Time Protocol (NTP): глобальная точность от 1 до 50 мс, в зависимости от расстояния между узлами

Очевидно, 1 – 50 мс – недостаточная степень точности для современных компьютеров, т.е невозможно иметь единое *абсолютное* время в каждом узле с достаточной точностью

К счастью, для алгоритмов управления оно и не нужно: достаточно в каждом узле иметь единое *относительное* время, т.е. одинаково упорядоченную последовательность событий – это показал Лэмпорт

Лесли Лэмпорт – основатель теории распределенных систем



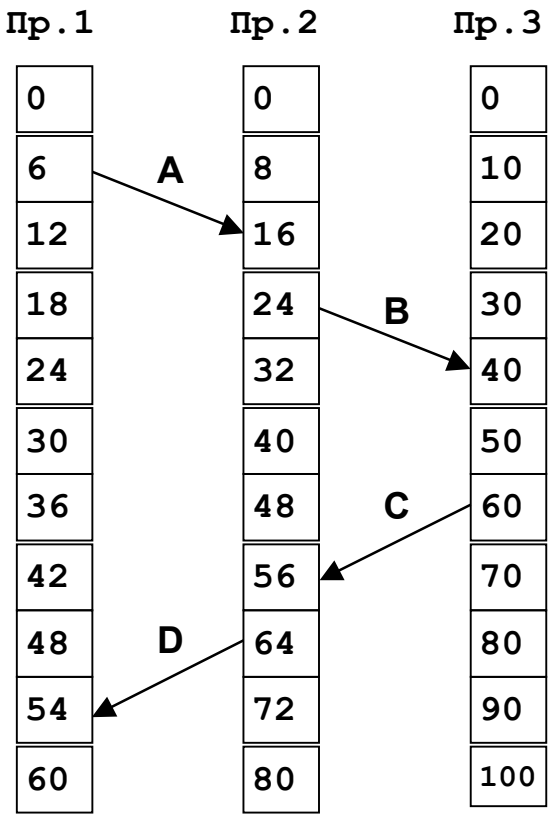
Базовая статья о методе временны**х** меток (timestamps):

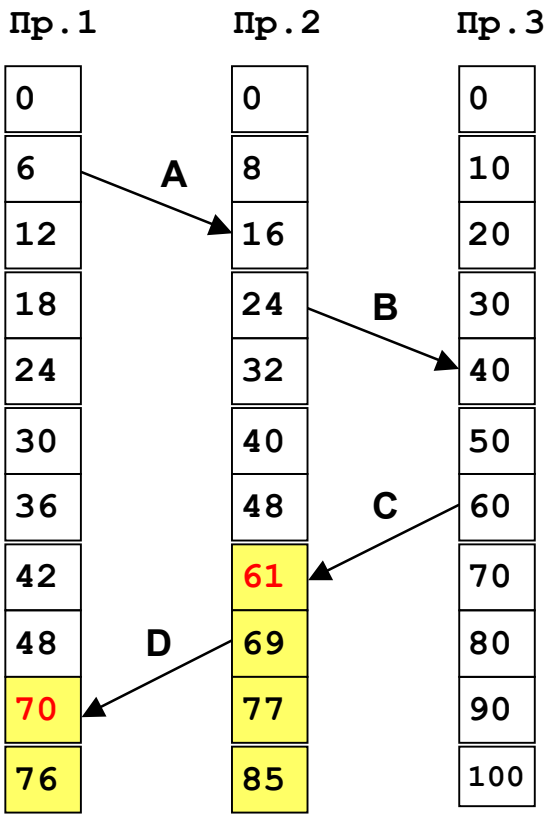
Leslie Lamport "Time, Clocks and the Ordering of Events in a Distributed System
Communications of the ACM **21** (7) (July 1978)

Другие работы:

- Алгоритм пекарни для взаимного исключения
- Множество распределенных алгоритмов: Global Snapshot, Byzantine Generals, Paxos (for consensus),…
- Основы темпоральной логики действий (TLA) для анализа параллельных программ
- LaTex – версия Tex'a

Идея метода временных меток (1)





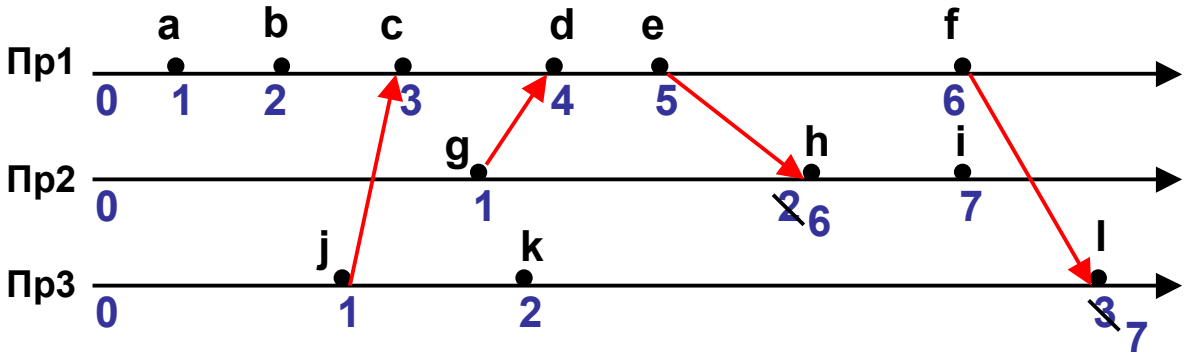
Три узла – три процесса, каждый с собственными часами;

A, B, C, D - сообщения

Подстройка часов по идее Лэмпорта

Идея метода временных меток (2)

- Глобальное физическое время нас не интересует; важно, чтобы все процессы системы договорились об одинаковом порядке относительного следования событий
- Глобально упорядочить можно только события, наблюдаемые извне каждого процесса, а именно, связанные с передачей сообщений
- Будем связывать «тики» часов с событиями в процессе, передавать показания часов вместе с сообщениями и соответственно подстраивать часы получателя
- Локальные события в процессах в периоды между подстройками часов считаем логически одновременными, или параллельными (concurrent)



a, b, c, … - события

1, 2, 3, … - показания локальных часов

Формализация логического времени

- В каждом процессе есть свои часы - счетчик логического времени
- Для синхронизации логических часов Лэмпорт определил отношение частичного порядка событий «произошло до»
  - Выражение $a \rightarrow b$ читается как «a произошло до b» и означает, что все процессы согласны, что сначала произошло событие «a», а затем «b»
- Это отношение очевидно в двух случаях:

  1) Если оба события произошли в одном процессе, ведь процесс – это *последовательность* событий

  2) Если событие «a» есть отправка сообщения одним процессом, а событие «b» - прием этого сообщения другим процессом
- Отношение $\rightarrow$ является транзитивным (но не рефлексивным)
- Если два события «x» и «y» произошли в различных процессах, которые не обменивались сообщениями, то отношения $x \rightarrow y$ и $y \rightarrow x$ являются неверными, а эти события называют параллельными (concurrent)
- Введем логическое время C (временн**ы**е метки) таким образом, что если $a \rightarrow b$, то $C(a) < C(b)$ (но обратное не обязательно истинно)

Идея метода временных меток (3)

- Логическое время (как и по часам) всегда идет вперед (C увеличивается), поэтому коррекция времени должна производиться только прибавлением к нему положительного значения и никогда – вычитанием
- Каждое сообщение маркируется временем его отправки
- Когда сообщение доставлено получателю, но часы получателя показывают время меньшее, чем время отправки, то получатель быстро подводит свои часы, чтобы они показывали на единицу большее время, чем время отправки
- Ни для каких a, b не должно выполняться C(a) = C(b), для этого:
  - в одном процессе: между любыми двумя событиями часы должны «протикать» минимум 1 раз
  - чтобы различать одновременные события, произошедшие в разных процессах, к метке времени добавляется постфикс – номер процесса-отправителя

Таким образом, однозначной глобальной временной характеристикой каждого сообщения является пара: его временная метка + номер узла-отправителя

Алгоритм метода временных меток

1. Часы Ci увеличивают свое значение с каждым событием в процессе Pi: Ci = Ci + 1
2. Если событие «a» есть посылка сообщения «m» процессом Pi, то в это сообщение вписывается временная метка tm=Ci(a)
3. В момент получения этого сообщения процессом Pj его время корректируется следующим образом: Cj = max(Cj, tm+1)

Принцип временного упорядочения на стороне приемника: сообщение X от узла i предшествует сообщению Y от узла k, если Ci < Ck, а при Ci = Ck – если i < k

Метки Лэмпорта дают множество частичных упорядочений событий, которые могут не совпадать для разных процессов из-за разных величин задержки передачи сообщений

Полное упорядочение событий в системе, одинаковое для всех процессов:

- если процессы посылают сообщения всем другим и получают подтверждения
- с событиями связываются *векторные* временные метки

Алгоритмы взаимного исключения

Централизованный

1. Центральный управляющий узел ведет очередь запросов на вход
2. Процесс, желающий войти в критический участок, посылает ему сообщение «запрос» и ждет, пока не получит сообщение «разрешение» (когда подходит его очередь)
3. Когда процесс выходит из критического участка, то посылает управляющему узлу сообщение «освобождение»

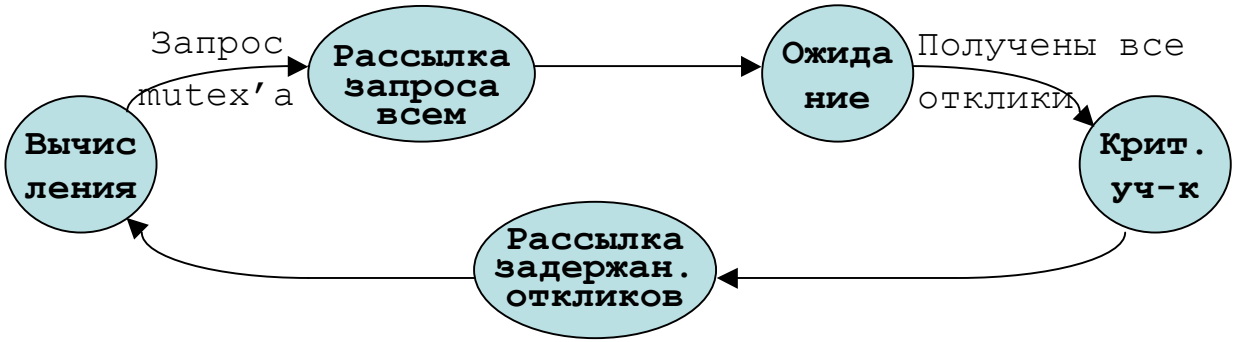
3 сообщения нужно послать для каждого случая доступа к критич. ресурсу

Распределенные алгоритмы

- С использованием временных меток
  - Распределенная очередь (Лэмпорт; Рикарт и Агравала)
  - Маркерный алгоритм
- Маркерное кольцо
- другие

Распределенный алгоритм взаимоисключений Рикарта и Агравала

- Когда процесс Pi хочет войти в критический участок, то рассылает всем сообщение «запрос» с текущей временной меткой
- Когда процесс Pk получает от Pj запрос войти в критический участок:
  - если он сам не посылал запрос, то посылает отклик
  - если он послал свой запрос, он сравнивает временные метки этих двух запросов и посылает отклик только если у его собственного запроса метка позже
  - в остальных случаях процесс Pk задерживает отправку отклика
- Процесс может войти в критический участок только после получения откликов ото всех других узлов сети
- После выхода из него он рассылает задержанные отклики на все ожидающие запросы



Обсуждение алгоритма Р-А

- Задержанные отклики в каждом процессе образуют «распределенную очередь» ожидания входа в критический участок
- Взаимное исключение. Процессы входят в критический участок в порядке врем. меток, вставленных в их запросы. К моменту входа в критический участок процесс Pj получил от всех других отклики, помеченные более поздним временем, чем его запрос – значит, нет более ранних запросов, чем запрос Pj.
- Отсутствие взаимоблокировок – благодаря тому, что порядок временных меток на всех узлах совпадает
- Отсутствие голодания: так как запросы обслуживаются в порядке их расположения в очереди, каждый процесс рано или поздно становится самым старым и будет обслужен
- Сложность: 2*(N – 1) сообщений на каждый вход в критич. участок, где N – число узлов:
  - (N – 1) запросов
  - (N – 1) откликов
- Отказ любого узла приводит к зависанию алгоритма; это устраняется таймированием ожидания отклика, причем очередь сохраняется (в отличие от централизованного случая), и работу алгоритма можно восстановить → значительное усложнение алгоритма

Маркерный алгоритм

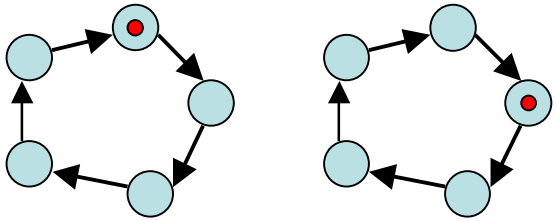
1. Маркер - массив M, где M[k] – временная метка: когда маркер последний раз находился в k-ом процессе
2. У каждого процесса – массив запросов R, элемент R[j] – временная метка сообщения - запроса от j-го процесса
3. Инициализация: маркер присваивается произвольному процессу
4. Перед входом в критический участок, если у процесса нет маркера, он рассылает всем другим запрос с временной меткой и ждет прихода маркера
5. После выхода из критического участка процесс Pj просматривает массив запросов в следующем порядке индексов: j+1, j+2, …, 1, 2, …, j -1, пока не найдет первый элемент R[k] такой, что R[k] >M[k], т.е. процесс Pk сделал запрос после последнего пребывания в нем маркера. Этому процессу Pk и посылается маркер.

(Вопрос 2)

Алгоритм “маркерное кольцо” (token ring)

- Все процессы в системе образуют логическое кольцо, независимо от реальной структуры сети
- По кольцу циркулирует маркер (token) - он переходит от процесса n к процессу n+1
- Когда процесс получает маркер, он анализирует ситуацию, не надо ли ему войти в критический участок:
  - если да, он оставляет маркер у себя и начинает выполнение критического участка программы, после выхода из него он передает маркер дальше по кольцу
  - если нет, он сразу же передает маркер дальше по кольцу

  т.е., если ни один из процессов не заинтересован во вхождении в критическую секцию, маркер просто циркулирует по кольцу



(Вопросы 3, 4)

Заключение

1. Распределенные системы – современная форма объединения вычислительных ресурсов для:
   - ❖ территориальной связи и удаленных вычислений
   - ❖ повышения:
     - ✓ производительности
     - ✓ надежности
     - ✓ масштабируемости
2. Стандарт программирования для кластеров – MPI – имеет другую модель параллелизма, чем OpenMP
3. Программирование для распределенных систем – сложнее, чем для сосредоточенных (в том числе многопроцессорных) из-за более сложных проблем синхронизации
4. Распределенное (т.е., децентрализованное) управление – решение задач управления системой всеми процессами сообща, без выделенного центрального процесса/узла – более надежное, чем централизованное
5. Распределенные алгоритмы (или протоколы) – предмет интенсивных исследований; актуален анализ их правильности

Вопросы

1. В каких ситуациях в параллельных алгоритмах удобно использовать барьеры?
2. Дайте неформальное обоснование полезных свойств этого алгоритма (взаимоисключение, отсутствие взаимоблокировок и голодания) и оцените, сколько всего сообщений требуется послать в системе из N узлов, чтобы процесс мог войти в свой критический участок.
3. То же для алгоритма «маркерное кольцо»
4. Как модифицировать этот алгоритм для случая ненадежной связи? - Связь с любым узлом может в любой момент прекратиться из-за отказа узла или линии связи.