Параллельные системы баз данных

1. Введение

1.1 Сверхбольшие базы данных

- Сверхбольшая база данных – это база данных, для которой время выполнения типичного запроса превышает допустимые для пользователя пределы.
- Решение – организовать параллельную обработку запроса на большом количестве процессорных узлов.

Области приложений сверхбольших баз данных

- электронная коммерция
- информационные системы здравоохранения
- геоинформационные системы
- электронные библиотеки
- мультимедиа архивы и коллекции
- научные базы данных
- …

Примеры систем сверхбольших баз данных

- база данных проекта *BaBar*
- база данных проекта EOS/DIS
- система SkyServer

База данных проекта *BaBar*

- Одной из самых больших научных баз данных является база данных проекта *BaBar*. Целью эксперимента *BaBar* является изучение поведения B-мезонов, получаемых на коллайдере PEP-II в Стэндфордском центре линейного ускорителя (Stanford Linear Accelerator Center). Детектор *BaBar* поставляет около 500 Гбайт информации ежедневно. Данная информация сохраняется в базе данных *BaBar*, объем которой сегодня составляет более 500 Тбайт. Система включает в себя 2000 процессоров и 100 серверов.

База данных проекта EOS/DIS

- Другим примером сверхбольшой базы данных является база данных проекта EOS/DIS (Earth Observation System/Data Information System), разрабатываемого агентством NASA в США. Система наблюдения земли EOS включает в себя множество спутников, которые собирают информацию, необходимую для изучения долгосрочных тенденций состояния атмосферы, океанов, земной поверхности. Начиная с 1998 года спутники поставляют информацию в объеме 1/3 петабайт (Petabyte - 10^{15} байт) в год. Предполагается, что к 2010 году общий объем поддерживаемых в системе данных превысит 20 петабайт.

Система SkyServer

- Еще одним примером системы, требующей обработки сверхбольших объемов данных, является система SkyServer проекта SDSS (Sloan Digital Sky Survey). Данный проект предполагает создание виртуальной обсерватории, доступной через Интернет. База данных проекта должна объединить в себе полную информацию о наблюдениях всех участков звездного неба различными обсерваториями мира. Начальный объем базы данных проекта оценивается в 40 терабайт. Работы по созданию виртуальной обсерватории ведутся также и в России.

1.2 Вводный пример

П (поставщики)

Код_П*	Имя_П	Город
23	Иванов И.И.	Москва
14	Петров П.П.	Самара

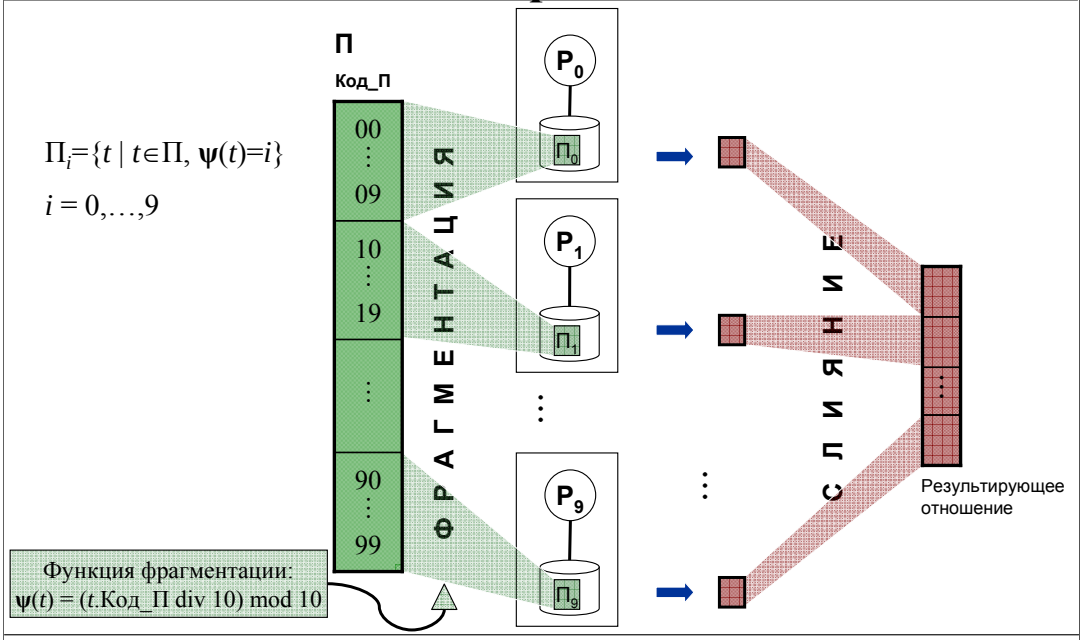
Д (Детали)

Код_Д*	Имя_Д	Цвет
3	Гайка	Красный
7	Болт	Синий

ПД (Поставки)

Код_ПД*	Код_П#	Код_Д#
1	14	7
2	23	3

Общая схема параллельной обработки запроса

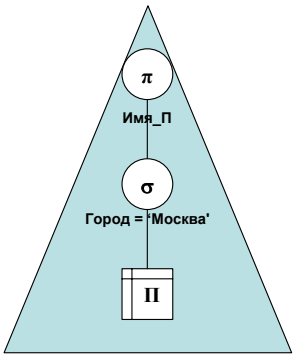


$\Pi_i = \{t \mid t \in \Pi, \psi(t) = i\}$

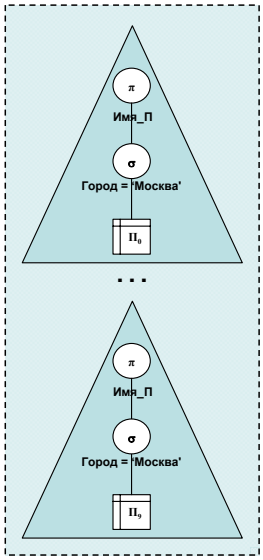
$i = 0,\ldots,9$

Параллельный план

```
/* Имена поставщиков,
проживающих в Москве */
SELECT Имя_П
FROM П
WHERE Город = 'Москва';
```

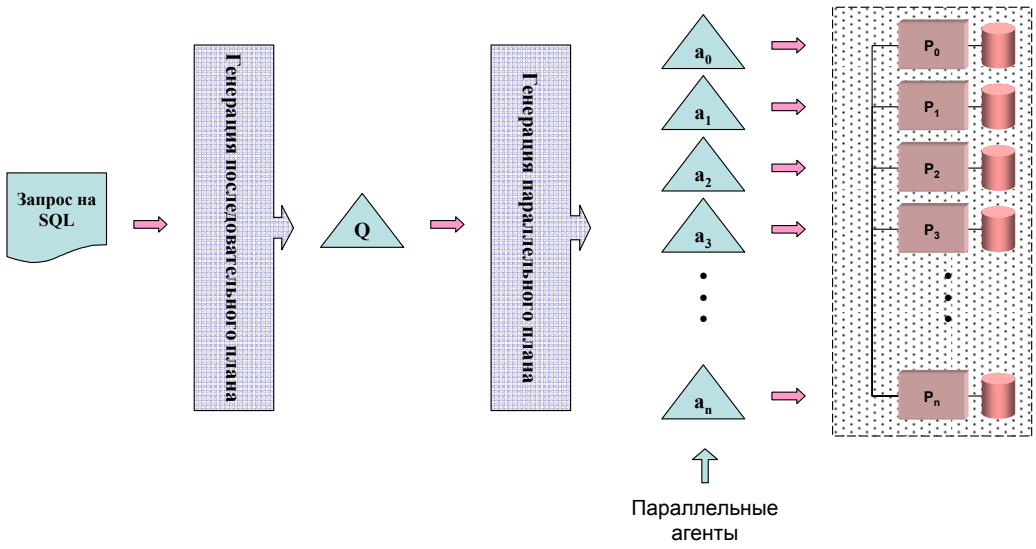


Последовательный план



Параллельный план

Общая схема обработки запроса



Ускорение!!!

- Два узла – ускорение в 2 раза!
- Десять узлов – ускорение в 10 раз!!!
- Тысяча узлов – ускорение в 1000 раз ?

Неприятные вещи

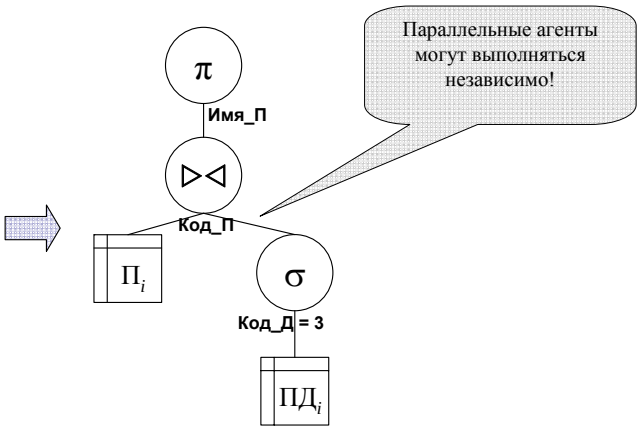
- Необходимость межпроцессорных обменов
- Перекосы в распределении загрузки
- Помехи (конкуренция за общие ресурсы)

Нет необходимости в межпроцессорных обменах

П фрагментировано по Код_П: $\psi_{П}(t) = (t.\text{Код\_П div } 10) \bmod 10$

ПД фрагментировано по Код_П : $\psi_{ПД}(t) = (t.\text{Код\_П div } 10) \bmod 10$

```
/* Имена поставщиков,
поставляющих
деталь с кодом 3 */
SELECT Имя_П
FROM П, ПД
WHERE П.Код_П = ПД.Код_П
AND ПД.Код_Д = 3;
```

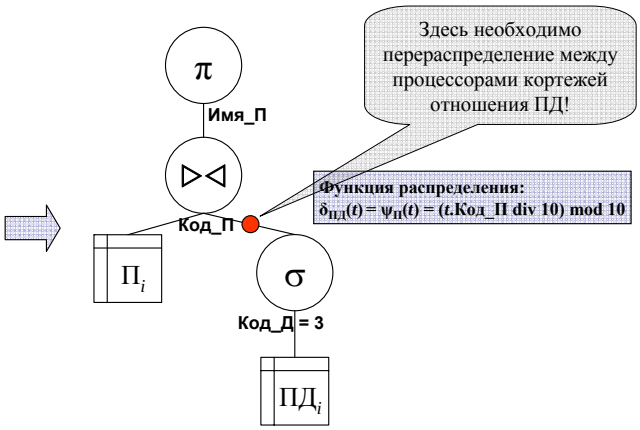


Есть необходимость в межпроцессорных обменах

П фрагментировано по Код_П: $\psi_{П}(t) = (t.Код\_П \text{ div } 10) \bmod 10$

ПД фрагментировано по Код_Д: $\psi_{ПД}(t) = (t.Код\_Д \text{ div } 10) \bmod 10$

```
/* Имена поставщиков,
поставляющих
деталь с кодом 3 */
SELECT Имя_П
FROM П, ПД
WHERE П.Код_П = ПД.Код_П
AND ПД.Код_Д = 3;
```



Перекосы в распределении загрузки

- Правило 80-20: 80% запросов адресуется к 20% базы данных

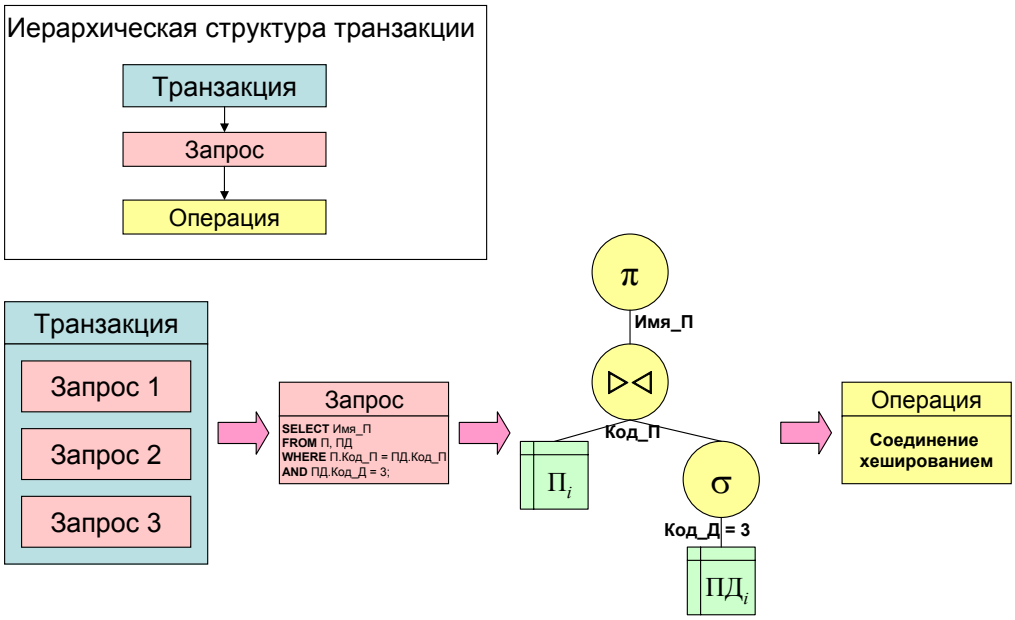
Помехи (конкуренция за общие ресурсы)

- Общая шина доступа к дискам
- Соединительная сеть

Параллельные системы баз данных

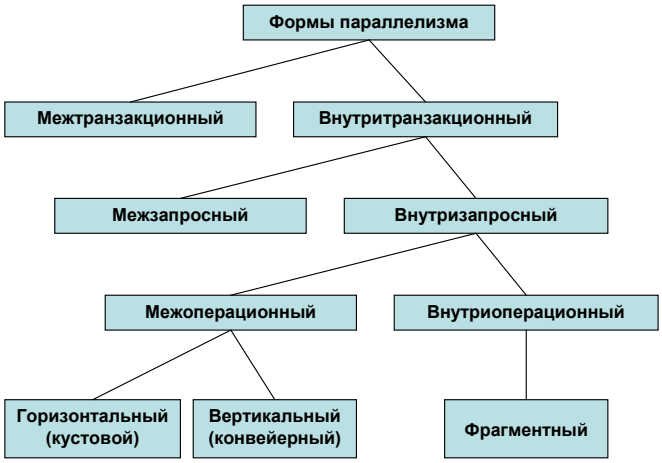
2. Формы параллельной обработки транзакций

2.1 Транзакция – запрос – операция

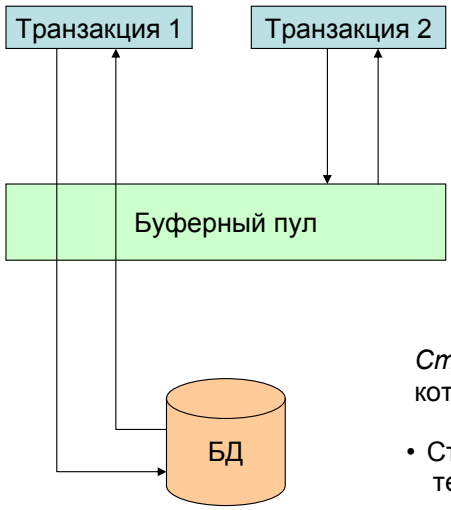


Параллельная обработка данных в том или ином виде присутствует во всех современных полнофункциональных СУБД. Это означает, что в некотором смысле все СУБД являются параллельными! Однако формы параллелизма в различных СУБД могут значительно отличаться друг от друга. В этой главе мы дадим классификацию и обзор основных форм параллельной обработки запросов над базой данных. При этом мы будем опираться на понятие *транзакции* SQL, представляющей собой последовательность операторов SQL, обладающую свойством атомарности относительно восстановления непротиворечивого состояния базы данных. Каждый SQL-оператор определяет некоторое действие над базой данных, которое мы будем называть *запросом*. Таким образом, мы можем считать, что транзакция состоит из последовательности запросов.

2.2 Формы параллелизма



2.3 Межтранзакционный параллелизм



Степень параллелизма – количество процессов, которые могут выполняться параллельно.

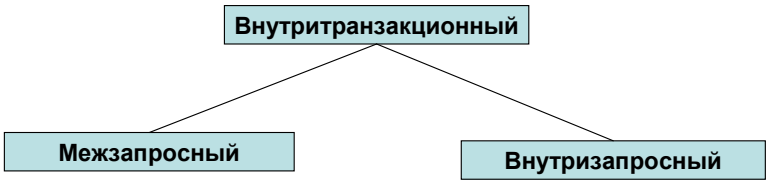
- Степень межтранзакционного параллелизма: теоретически неограниченна!

Межтранзакционный параллелизм подразумевает параллельное выполнение множества независимых транзакций над одной и той же базой данных. Данный вид параллелизма присутствует уже в однопроцессорных системах в виде так называемого многозадачного режима и основан на перекрытии задержек ввода-вывода. Пока одна транзакция ожидает завершения обмена с диском, другая транзакция может выполнять операции с данными, размещенными в оперативной памяти.

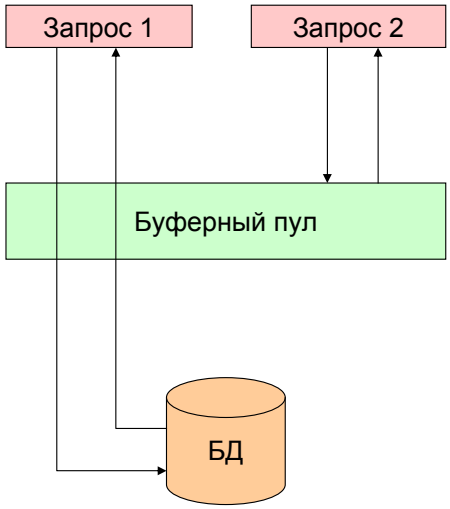
Межтранзакционный параллелизм позволяет существенно увеличить суммарную производительность системы баз данных в режиме оперативной обработки транзакций (OLTP). Режим OLTP характеризуется наличием большого количества одновременно выполняемых коротких транзакций, что свойственно для бизнес-приложений систем баз данных.

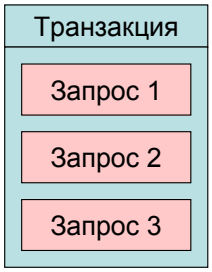
Межтранзакционный параллелизм также должен поддерживаться и в мультипроцессорных системах (в рамках одного процессора), так как в противном случае мы получим очень плохое соотношение цена-производительность для режима OLTP.

2.4 Внутритранзакционный параллелизм



2.5 Межзапросный параллелизм

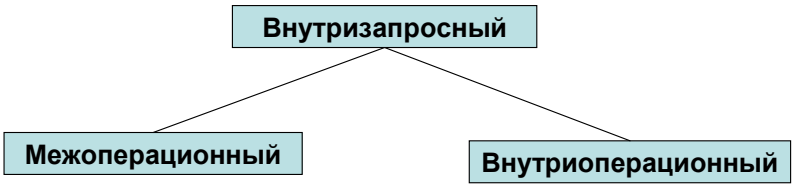




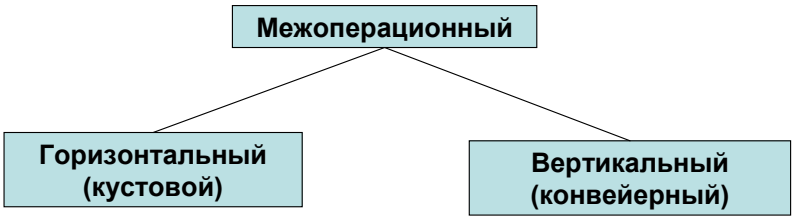
- Степень межзапросного параллелизма ограничена:
  1. Количеством SQL-операторов в транзакции
  2. Ограничениями предшествования между отдельными SQL-операторами
- Межзапросный параллелизм не используется в СУБД

Межзапросный параллелизм предполагает параллельное выполнение запросов, соответствующих различным SQL-операторам одной и той же транзакции. Степень межзапросного параллелизма, однако, ограничена как количеством SQL-операторов, составляющих данную транзакцию, так и ограничениями предшествования между отдельными SQL-операторами. Межзапросный параллелизм не поддерживается большинством современных СУБД, так как это потребовало бы от программиста явной спецификации межзапросных зависимостей с помощью некоторых специальных языковых конструкций.

2.6 Внутризапросный параллелизм



2.7 Межоперационный параллелизм



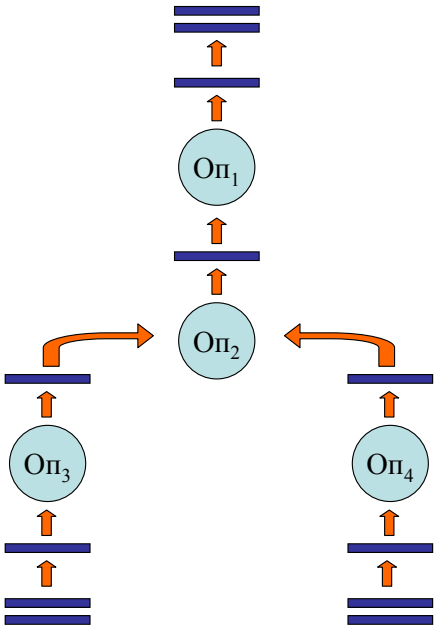
2.8 Кустовой параллелизм

Кустовой (*горизонтальный*) *параллелизм* предполагает параллельное выполнение независимых поддеревьев дерева, представляющего план выполнения запроса. Основная проблема, связанная с кустовым параллелизмом, заключается в том, что очень трудно обеспечить, чтобы два подплана одного плана начали генерировать выходные данные в правильное время и в правильном темпе. При этом правильное время далеко не всегда означает одинаковое время (например, для входных потоков операции хеш-соединения), а правильный темп далеко не всегда означает одинаковый темп (например, для случая, когда входные потоки соединения слиянием имеют различные размеры). В силу указанных причин кустовой параллелизм редко используется на практике.

2.9 Конвейерный параллелизм

- Синхронный конвейер
- Асинхронный конвейер

Синхронный конвейер



Основным недостатком синхронного конвейера является блокирующий характер операций конвейерной обработки отдельных гранул. Если некоторая операция задерживается с обработкой очередной гранулы данных, то она блокирует работу всего конвейера. Для преодоления указанного недостатка может быть использован *асинхронный конвейер*, в котором поставщик и потребитель работают независимо друг от друга, а данные передаются через некоторый буфер. Поставщик помещает производимые гранулы в буфер, а потребитель забирает гранулы из данного буфера в соответствующем порядке. При этом необходимо определенное управление потоком данных, которое препятствовало бы переполнению указанного буфера в случае, когда потребитель работает медленнее, чем поставщик.

Степень конвейерного параллелизма

- Степень конвейерного параллелизма ограничена: количеством операций, вовлекаемых в конвейер.

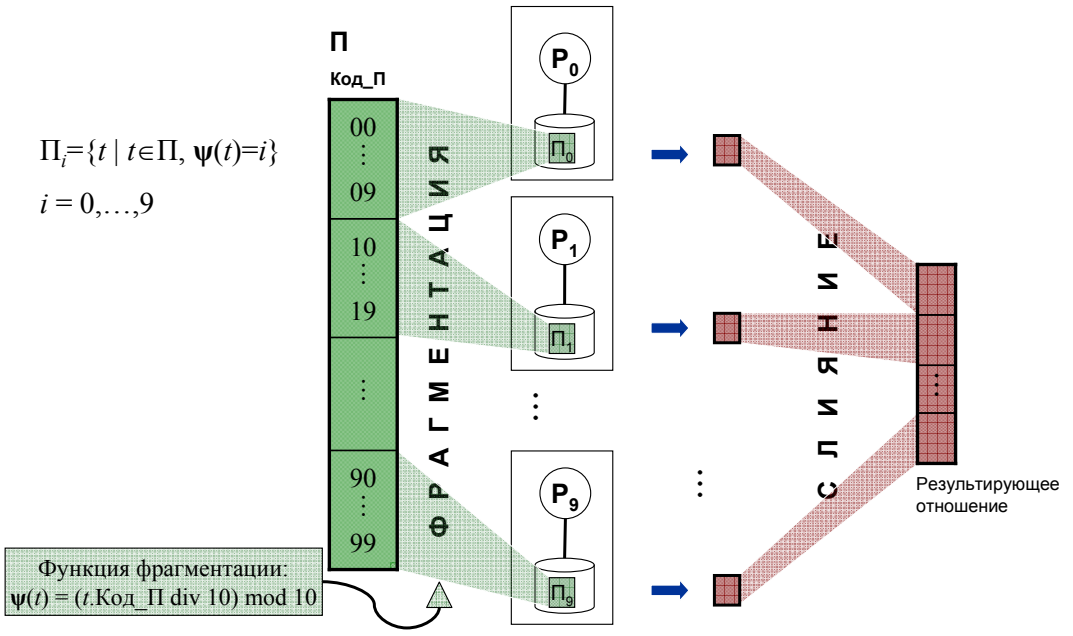
- Для реляционных систем баз данных длина конвейера редко превышает 10 операций.

2.10 Внутриоперационный параллелизм

Внутриоперационный параллелизм реализуется в основном в форме *фрагментного параллелизма*, предполагающего разбиение входных отношений на фрагменты, которые могут обрабатываться независимо на разных узлах многопроцессорной системы.

Мы можем рассматривать и другие формы внутриоперационного параллелизма, базирующиеся на делении реляционной операции на подоперации. Однако данные формы параллелизма концептуально ничем не отличаются от рассмотренных выше и на практике большого значения не имеют.

2.11 Фрагментный параллелизм



Фрагментный параллелизм предполагает *фрагментацию* (разбиение на непересекающиеся части) отношения, являющегося аргументом реляционной операции. Одиночная реляционная операция выполняется в виде нескольких параллельных процессов (*агентов*), каждый из которых обрабатывает отдельный фрагмент отношения. Получаемые результирующие фрагменты сливаются в общее результирующее отношение.

Фрагментация

В реляционных системах баз данных фрагментация подразделяется на *вертикальную* и *горизонтальную*. *Вертикальная фрагментация* подразумевает разбиение отношения на фрагменты по столбцам (атрибутам). *Горизонтальная фрагментация* подразумевает разбиение отношения на фрагменты по строкам (кортежам). Практически все параллельные СУБД, поддерживающие фрагментный параллелизм, используют только горизонтальную фрагментацию. Поэтому везде ниже мы будем рассматривать только горизонтальную фрагментацию.

Степень фрагментного параллелизма

- Теоретически фрагментный параллелизм способен обеспечить сколь угодно высокую степень распараллеливания реляционных операций. Однако на практике степень фрагментного параллелизма может быть существенно ограничена следующими двумя факторами.
  - Во-первых, фрагментация отношения может зависеть от семантики операции. Например, операция соединения одних и тех же отношений по разным атрибутам требует различной фрагментации. Однако повторное разбиение фрагментированного отношения на новые фрагменты и распределение полученных фрагментов по процессорным узлам может быть связано с очень большими накладными расходами.
  - Во-вторых, перекосы в распределении значений атрибутов фрагментации могут привести к значительным перекосам в размерах фрагментов и, как следствие, к существенному дисбалансу в загрузке процессоров.

Параллельные системы баз данных

3. Определение параллельной системы баз данных

Определение параллельной системы баз данных

- Неформальное определение
- Формальное определение в виде виртуальной машины

3.1 Неформальное определение параллельной системы баз данных

- ***Параллельная система баз данных*** - это СУБД, реализованная на многопроцессорной системе с высокой степенью связности.
- ***Многопроцессорная система с высокой степенью связности*** это - система, включающая в себя много процессоров и много дисков, в которой процессоры объединены между собой быстродействующей соединительной сетью, причем *время обмена данными по сети сравнимо с временем обмена данными с локальным диском*.

Приведенное определение исключает из рассмотрения распределенные СУБД, реализуемые на нескольких независимых компьютерах, объединенных локальной и (или) глобальной сетями, для которых характерны свои специфические проблемы, такие как географическая удаленность, локальная автономность, программная и аппаратная гетерогенность, и применительно к которым не рассматривается комплекс важных проблем, связанных с большим количеством процессорных узлов. Однако данное определения включает в себя широкий спектр систем, начиная с традиционных последовательных СУБД, портированных на симметричные мультипроцессорные системы и использующих только межтранзакционный параллелизм, и кончая сложными параллельными системами, реализованными на кластерах или мультипроцессорах с массовым параллелизмом, использующими фрагментный параллелизм.

3.2 Виртуальная параллельная машина баз данных

- виртуальные процессоры
- виртуальные модули памяти
- виртуальные диски
- виртуальная соединительная сеть

Виртуальная параллельная машина баз данных строится из следующих виртуальных устройств: *виртуальные процессоры*, *виртуальные модули памяти*, *виртуальные диски, виртуальная соединительная сеть*.

Виртуальный процессор

- *Виртуальный процессор* - это виртуальное устройство, используемое для выполнения отдельной задачи, определяемой как *процесс базы данных.*
- *Процесс базы данных* – реализация запроса (отдельного оператора SQL).
- *Физическое представление*:
  - микропроцессор
  - процессорное ядро
  - процессорный модуль (несколько микропроцессоров на общей памяти)
  - процесс операционной системы (на одном микропроцессоре может одновременно выполняться несколько процессов операционной системы)

Виртуальный процессор - это виртуальное устройство, используемое для выполнения отдельной задачи, определяемой как *процесс базы данных*. Типичным примером процесса базы данных является запрос или агент запроса (при использовании фрагментного параллелизма). В реальной системе виртуальный процессор может быть представлен процессорным ядром, микропроцессором или процессорным модулем. Если при этом на одном физическом процессоре выполняется несколько процессов базы данных в режиме разделения времени, то мы вправе говорить, что данный физический процессор реализует несколько виртуальных процессоров.

Виртуальный модуль памяти

- *Виртуальный модуль памяти* - это виртуальное устройство, используемое для буферизации объектов базы данных.
  - Виртуальный процессор не может получить доступ к объекту базы данных иначе, чем через его образ, загруженный в некоторый доступный ему виртуальный модуль памяти.
- Каждый виртуальный процессор должен быть соединен с некоторым модулем виртуальной памяти.
- Количество виртуальных модулей памяти в виртуальной параллельной машине баз данных не превосходит количество виртуальных процессоров.
- *Физическое представление*: совокупность физических модулей оперативной памяти процессорного модуля.

Виртуальный модуль памяти - это виртуальное устройство, используемое для буферизации объектов базы данных. Типичным примером объекта реляционной базы данных является отношение или его фрагмент (при использовании фрагментного параллелизма). Виртуальный процессор не может получить доступ к объекту базы данных иначе, чем через его образ, загруженный в некоторый доступный ему виртуальный модуль памяти. В соответствие с этим мы можем всегда считать, что количество виртуальных модулей памяти в виртуальной параллельной машине баз данных не превосходит количество виртуальных процессоров. В реальной системе виртуальный модуль памяти обычно реализуется в виде физических модулей оперативной памяти. При этом один физический модуль памяти может реализовывать несколько виртуальных модулей памяти, и несколько физических модулей памяти могут реализовывать один виртуальный модуль памяти.

Виртуальный диск

- *Виртуальный диск* - это виртуальное устройство, используемое для хранения объектов базы данных.
- Виртуальный процессор не может соединяться напрямую с виртуальным диском.
- *Физическое представление*:
  - Жесткий диск (винчестер)
    - HDD
      - Hard Disk Drive
    - НЖМД
      - Накопитель на жестком магнитном диске
  - Дисковый массив
    - RAID
      - Redundant Arrays of Inexpensive Disks
      - Redundant Arrays of Independent Disks

Виртуальный диск - это виртуальное устройство, используемое для хранения объектов базы данных. В реальной системе виртуальный диск обычно реализуется в виде физического дискового устройства или дискового массива.

N

Виртуальная соединительная

- *Виртуальная соединительная сеть* - это виртуальное устройство, обеспечивающее передачу объектов базы данных в виде сообщений переменной длины из одного виртуального модуля памяти в другой.
- В виртуальной параллельной машине баз данных может существовать не более одной виртуальной соединительной сети.
- Если в виртуальной машине имеется только один модуль памяти, то виртуальная сеть должна отсутствовать.
- С виртуальной соединительной сетью могут соединяться дугами только виртуальные процессоры
  - Передача сообщения в виртуальной параллельной машине баз данных может осуществляться только посредством коммуникативных действий, выполняемых каким-либо виртуальным процессором, соединенным с соответствующим виртуальным модулем памяти).
- Допускаются конфигурации, в которых не каждый виртуальный процессор соединен с виртуальной сетью.
- *Физическое представление*: сетевые коммутаторы и маршрутизаторы.

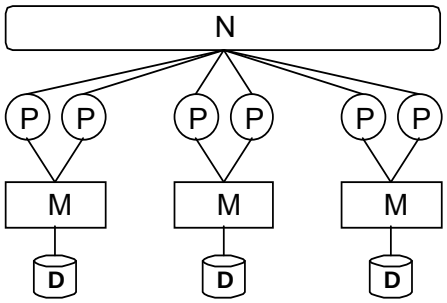
Виртуальная соединительная сеть - это виртуальное устройство, обеспечивающее передачу объектов базы данных в виде сообщений переменной длины из одного виртуального модуля памяти в другой.

Так как мы исключаем из рассмотрения распределенные системы и предполагаем, что передача сообщений между любыми двумя узлами сети осуществляется примерно за одинаковое время, то без существенного ограничения общности мы можем полагать, что в виртуальной параллельной машине баз данных может существовать не более одной виртуальной соединительной сети. При этом следует отметить, если в виртуальной машине имеется только один модуль памяти, то виртуальная сеть должна отсутствовать.

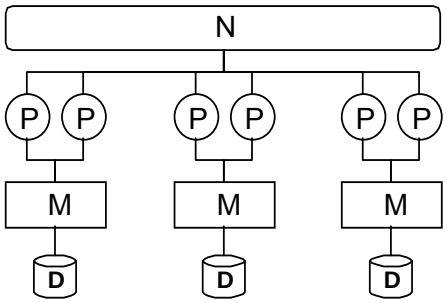
Передача сообщения в виртуальной параллельной машине баз данных может осуществляться только посредством коммуникативных действий, выполняемых каким-либо виртуальным процессором, соединенным с соответствующим виртуальным модулем памяти. Поэтому с виртуальной соединительной сетью могут соединяться дугами только виртуальные процессоры. При этом мы допускаем существование конфигураций, в которых не каждый виртуальный процессор соединен с виртуальной сетью.

Определение виртуальной параллельной машины баз данных

Виртуальная параллельная машина баз данных – это связный граф, узлы которого соответствуют различным виртуальным устройствам, а ребра соответствуют потокам данных.

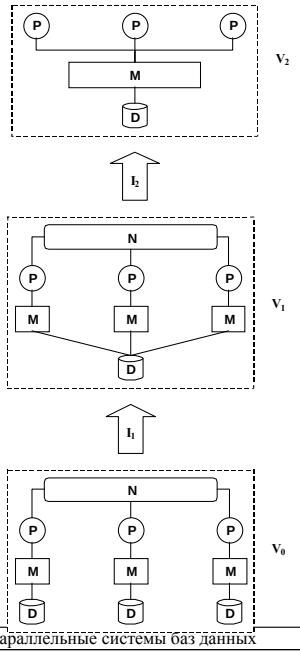


(а) С прямыми ребрами



(б) С ломаными ребрами

Иерархия виртуальных машин параллельных баз данных (пример гибридной архитектуры)



- **V0** имеет аппаратно-программную реализацию
  - объединение в сеть нескольких самостоятельных компьютеров на базе MPI.
- **I1** *реализует* виртуальную машину **V1**, на базе **V0**.
  - **I1** –комплекс программ для **V0**, реализующий *виртуальный диск*, объединяющий в себе дисковое пространство всех физических дисков системы.
- **I2** *реализует* виртуальную машину **V2**, на базе **V1**.
  - **I2** –комплекс программ для **V1**, реализующий общую *виртуальную память*, объединяющую в себе свободное адресное пространство всех физических модулей памяти.

Следует отметить, что виртуальная машина баз данных представляет собой не что иное, как некоторый уровень абстракции в системной иерархии программных модулей СУБД и операционной системы, реализующих систему баз данных на конкретной аппаратной

платформе. При этом в рамках одной системы мы можем рассматривать несколько таких уровней абстракции. Виртуальная машина каждого следующего уровня реализуется на основе функций и сервисов, предоставляемых виртуальной машиной предыдущего уровня.

Параллельные системы баз данных

4. Классификация многопроцессорных систем

Таксономии

- Классификация Флинна
- Структурно-функциональная классификация

3.1. Классификация Флинна

	Single Instruction	Multiple Instruction
Single Data	**SISD**	**MISD**
Multiple Data	**SIMD**	**MIMD**

SISD machines: These are the conventional systems that contain one CPU and hence can accommodate one instruction stream that is executed serially. Nowadays many large mainframes may have more than one CPU but each of these execute instruction streams that are unrelated. Therefore, such systems still should be regarded as (a couple of) SISD machines acting on different data spaces. Examples of SISD machines are for instance most workstations like those of DEC, Hewlett-Packard, IBM and SGI. The definition of SISD machines is given here for completeness' sake. We will not discuss this type of machines in this report.

SIMD machines: Such systems often have a large number of processing units, ranging from 1,024 to 16,384 that all may execute the same instruction on different data in lock-step. So, a single instruction manipulates many data items in parallel. Examples of SIMD machines in this class are the CPP DAP Gamma II and the Quadrics Apemille which are not marketed anymore since about 2 years. Nevertheless, the concept is still interesting and it may be expected that this type of system will come up again or at least as a co-processor in large, heterogeneous HPC systems.

Another subclass of the SIMD systems are the vectorprocessors. Vectorprocessors act on arrays of similar data rather than on single data items using specially structured CPUs. When data can be manipulated by these vector units, results can be delivered with a rate of one, two and — in special cases — of three per clock cycle (a clock cycle being defined as the basic internal unit of time for the system). So, vector processors execute on their data in an almost parallel way but only when executing in vector mode. In this case they are several times faster than when executing in conventional scalar mode. For practical purposes vectorprocessors are therefore mostly regarded as SIMD machines. An example of such a system is for instance the NEC SX-8B.

MISD machines: Theoretically in these types of machines multiple instructions should act on a single stream of data. As yet no practical machine in this class has been constructed nor are such systems easily to conceive. We will disregard them in the following discussions.

MIMD machines: These machines execute several instruction streams in parallel on different data. The difference with the multi-processor SISD machines mentioned above lies in the fact that the instructions and data are related because they represent different parts of the same task to be executed. So, MIMD systems may run many sub-tasks in parallel in order to shorten the time-to-solution for the main task to be executed. There is a large variety of MIMD systems and especially in this class the Flynn taxonomy proves to be not fully adequate for the classification of systems. Systems that behave very differently like a four-processor NEC SX-8 and a thousand processor IBM p690 fall both in this class. In the following we will make another important distinction between classes of systems and treat them accordingly.

3.2. Структурно-функциональная классификация

- *SMP* (Symmetric MultiProcessor) - симметричные мультипроцессорные системы
- *NUMA* (Non-Uniform Memory Architecture) - мультипроцессорные системы с неоднородным доступом к памяти
- *MPP* (Massively Parallel Processing) - массово параллельные системы
- *Clusters* (кластеры) - наборы полноценных компьютеров, объединенных в единую вычислительную систему
- *Constellations* (созвездия) – ассоциации мультипроцессоров

В основе структурно-функциональной классификации лежат структурные различия в компоновке различных функциональных элементов, основными из которых являются процессоры и оперативная память.

SMP

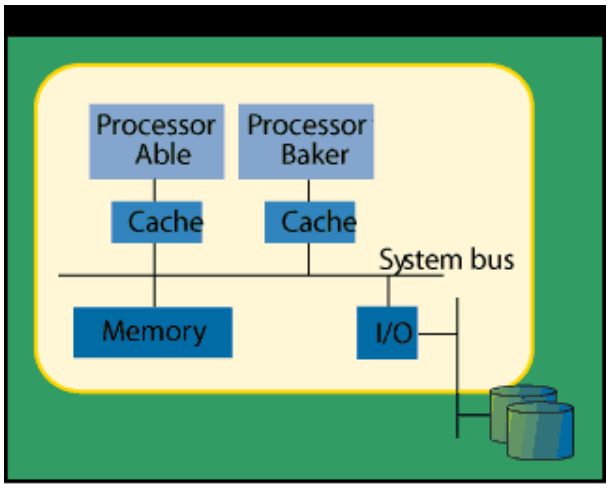
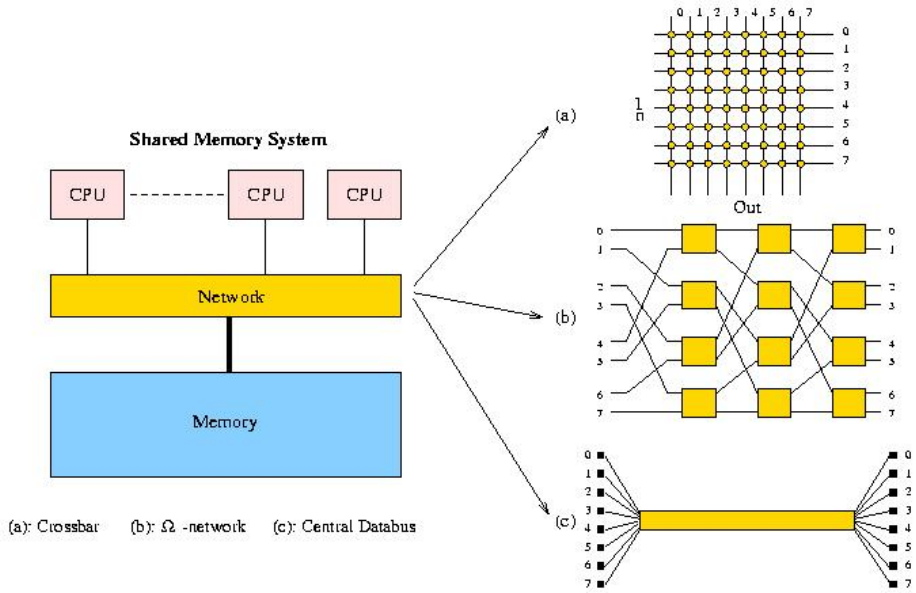


FIGURE 1. Symmetric multiprocessor.

SMP архитектуры представляют собой многопроцессорные системы, в которых все процессоры соединены посредством шины или коммутатора типа *crossbar* с общей оперативной памятью. В таких системах время обращения к физически единой общей памяти одинаково для всех процессоров. Поэтому SMP системы относят к классу *UMA* (*Uniform Memory Architecture*). Узким местом ранних SMP систем являлась общая шина доступа к оперативной памяти, унаследованная ими от мини-ЭВМ. В силу этого количество процессоров в них редко превышало 32. Во второй половине 90-х годов прошлого века в SMP системах шину заменили системным коммутатором. С его помощью оказалось возможно повысить число процессоров сначала до нескольких десятков, а потом и до сотен. Вместе с тем следует отметить, что SMP системы имеют относительно высокую стоимость.

SMP: Interconnect



NUMA архитектура

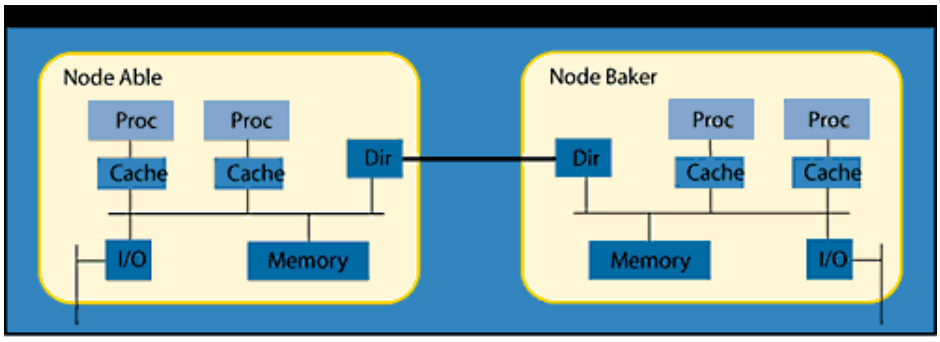
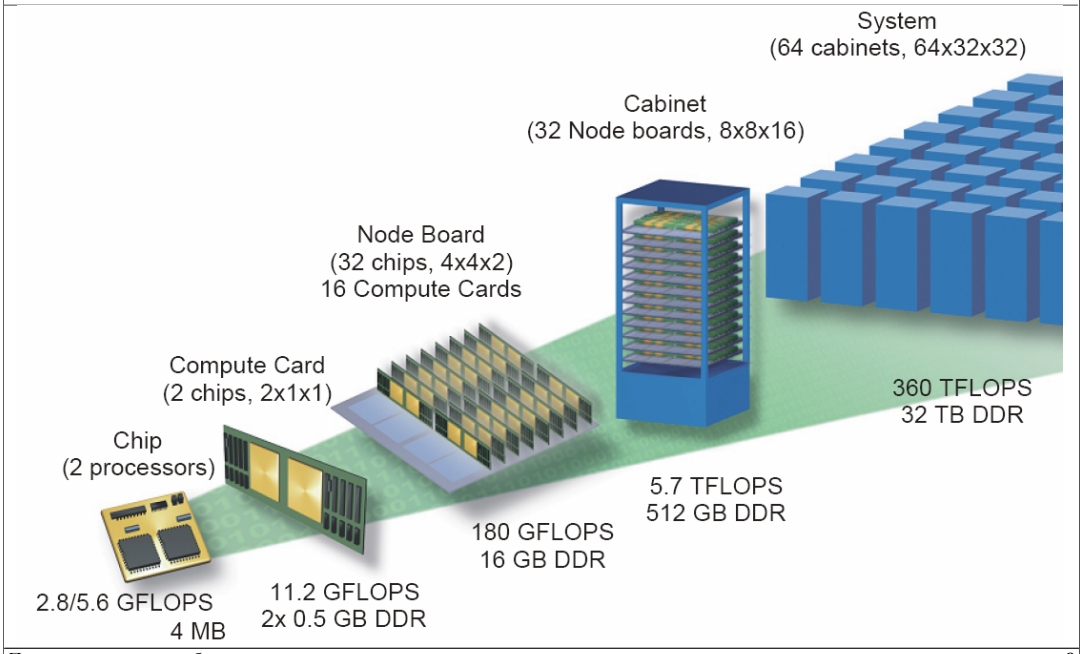


FIGURE 2. NUMA organization

NUMA архитектуры характеризуются наличием физически распределенной оперативной памяти, разделяемой всеми процессорными модулями в качестве единого адресного пространства. При этом отдельный процессорный модуль может представлять собой SMP систему с общей шиной. Процессорные модули NUMA систем соединяются сетью. При этом для обеспечения унифицированного доступа к разделяемой памяти и синхронизации изменений в кэш-памяти процессоров в каждом процессорном модуле используется специальное устройство Dir. Системы с NUMA архитектурой могут включать в себя десятки процессорных модулей с общим числом процессоров более тысячи. Однако неоднородность доступа к разделяемой оперативной памяти ведет к тому, что при выборке данных из памяти чужого модуля мы будем наблюдать замедление от 200% до 750% в различных NUMA системах, и эта цифра будет расти с увеличением числа процессорных модулей. Следует заметить, что разница во времени доступа к локальной и удаленной кэш-памяти в различных NUMA системах составляет 400-1500%. Поэтому производительность NUMA систем может существенным образом снижаться на приложения, требующих частой синхронизации данных. При этом NUMA системы характеризуются достаточно высокой стоимостью. В настоящее время NUMA системы практически исчезли с рынка.

MPP (Massive parallel processing)



MPP архитектуры строятся как совокупность большого числа однородных процессорных модулей, соединенных специальной высокоскоростной сетью. При этом каждый модуль имеет собственную оперативную память и, возможно, собственные диски. Подобные системы могут включать в себя тысячи процессорных устройств. Ключевой проблемой при использовании MPP систем является такое распределение данных между процессорными модулями, которое позволило бы разделить задачу на подзадачи (по одной на каждый модуль), свести к минимуму обмены между различными подзадачами и обеспечить сбалансированную загрузку всех процессорных модулей. Во многих случаях это является в высшей степени нетривиальной задачей. Следует отметить, что современные тенденции в развитии аппаратного обеспечения привели к фактическому слиянию MPP и кластерных архитектур.

Кластер (cluster)

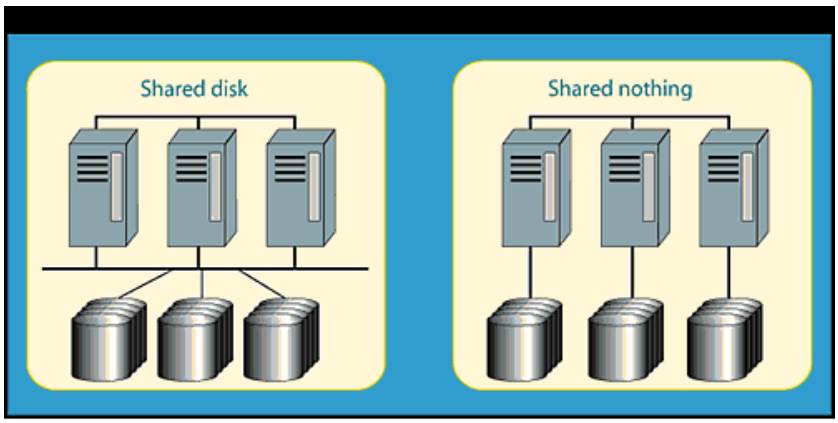
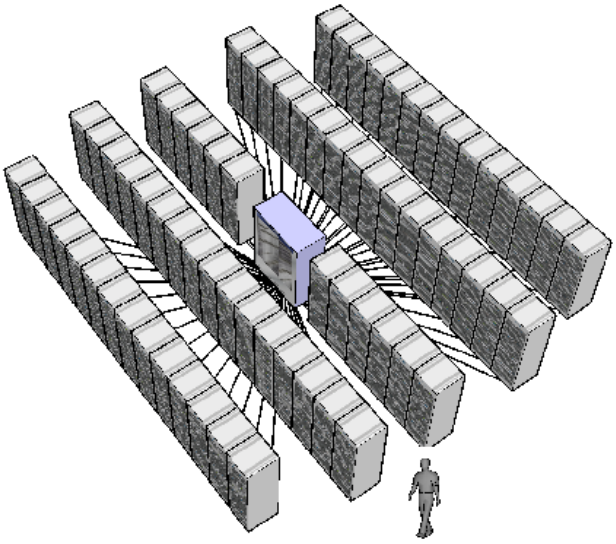


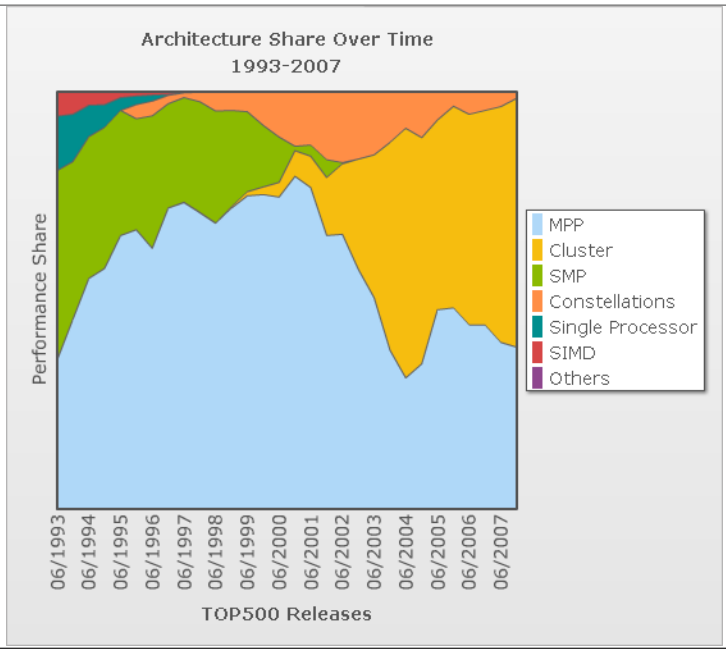
FIGURE 3. Two cluster organizations.

Кластер - это связанный сетью набор полноценных компьютеров, используемый в качестве единого вычислительного ресурса. В общем случае кластерная система может включать в себя дисковые массивы и другие специализированные устройства. В качестве узлов кластера могут использоваться как однопроцессорные системы, так и системы с архитектурой SMP на общей шине. Кластеры сочетают в себе экономичность и высокую вычислительную мощность и претендуют на доминирующее положение в области высокопроизводительных вычислений. Тем не менее следует заметить, что для кластеров характерны те же общие проблемы с распределением данных и балансировкой загрузки, что и для MPP систем.

Constellation (созвездие)

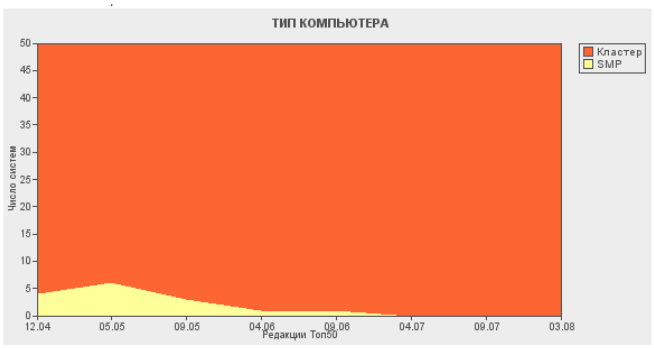


www.top500.org (ноябрь 2007)



TOP50 СНГ (8-ая редакция от 27.03.2008)
http://supercomputers.ru

TOP50 СНГ (8-ая редакция от 27.03.2008)
http://supercomputers.ru



Тип компьютера	1-ая редакция	2-ая редакция	3-я редакция	4-ая редакция	5-ая редакция	6-ая редакция	7-ая редакция	8-ая редакция
Кластер	46	44	47	49	49	50	50	50
SMP	4	6	3	1	1	0	0	0

Изменения:

Тип компьютера	1-ая редакция	2-ая редакция	3-я редакция	4-ая редакция	5-ая редакция	6-ая редакция	7-ая редакция	8-ая редакция
Кластер	46	-2	+3	+2	0	+1	0	0
SMP	4	+2	-3	-2	0	-1	0	0

Параллельные системы баз данных

5. Архитектура параллельных систем баз данных

Содержание

5.1. Классификация

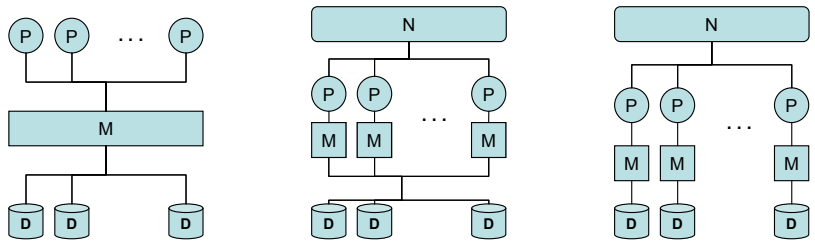
- Классификация Стоунбрейкера
- Виртуально-иерархическая классификация

Классификация Стоунбрейкера

- *SE* (*Shared-Everything*) - архитектура с разделяемыми памятью и дисками
- *SD* (*Shared-Disks*) - архитектура с разделяемыми дисками
- *SN* (*Shared-Nothing*) - архитектура без совместного использования ресурсов

Наиболее распространенной системой классификации параллельных систем баз данных является система, предложенная Майклом Стоунбрейкером (Michael Stonebraker) в работе [Stonebraker 86].

Классификация Стоунбрейкера



(а) SE (б) SD (в) SN

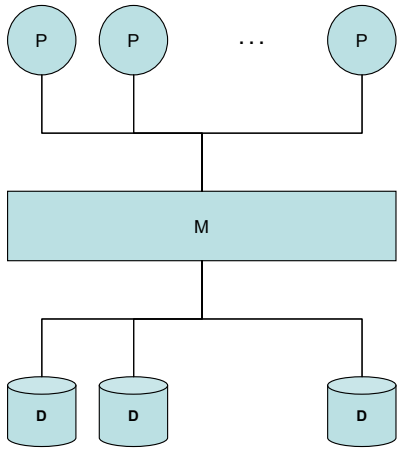
В соответствие с классификацией Стоунбрейкера параллельные системы баз данных могут быть разделены на следующие три базовых класса в зависимости от способа разделения аппаратных ресурсов:

1) *SE* (*Shared-Everything*) - архитектура с разделяемыми памятью и дисками [Рис. 1 (а)];

2) *SD* (*Shared-Disks*) - архитектура с разделяемыми дисками [Рис. 1 (б)];

3) *SN* (*Shared-Nothing*) - архитектура без совместного использования ресурсов [Рис. 1 (в)].

SE (*Shared-Everything*) - архитектура с разделяемыми памятью и дисками



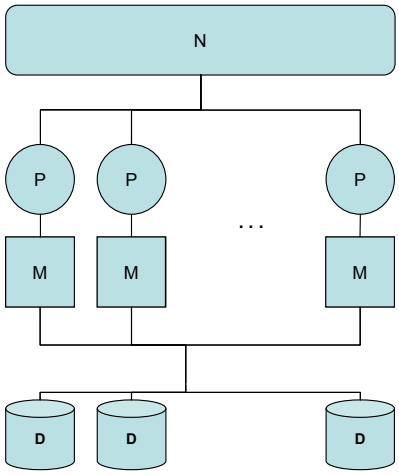
- Все процессоры разделяют общую оперативную память.
- Все диски напрямую доступны всем процессорам с одинаковым временем доступа.
- Межпроцессорные коммуникации осуществляются через общую оперативную память.
- Каждый процессор в SE системе имеет собственную кэш-память.
- Аппаратная платформа: SMP

SE архитектура (в работе [Stonebraker 86] эта архитектура обозначается как *Shared-Memory*) представляет системы баз данных, в которых все диски напрямую доступны всем процессорам с одинаковым временем доступа и все процессоры разделяют общую оперативную память [Рис. 1 (a)]. Межпроцессорные коммуникации в *SE* системах осуществляются через общую оперативную память. Доступ к дискам в *SE* системах обычно осуществляется через общий буферный пул. При этом следует отметить, что каждый процессор в *SE* системе имеет собственную кэш-память.

Существует большое количество параллельных систем баз данных с *SE* архитектурой. По существу все ведущие коммерческие СУБД сегодня имеют реализацию на базе *SE* архитектуры. В качестве одного из первых примеров портирования с однопроцессорной системы на *SE* архитектуру можно привести реализацию DB2 на IBM3090 с 6 процессорами [Cheng 84]. Другим примером является параллельное построение индексов в Informix OnLine 6.0 [Davison 92]. Следует отметить, однако, что подавляющее большинство коммерческих *SE* систем использует только межтранзакционный параллелизм (то есть внутритранзакционный параллелизм отсутствует). Тем не менее, к настоящему моменту создано несколько исследовательских прототипов *SE* систем, использующих внутризапросный параллелизм, например, XPRS [Stonebraker 88], DBS3 [Bergsten 91, Bergsten 93a] и Volcano [Graefe 90, Graefe 94].

Базовой аппаратной платформой для реализации систем с *SE* архитектурой обычно служит SMP, хотя потенциально *SE* системы можно строить на платформах с архитектурой NUMA и даже MPP с виртуально общей, физически распределенной памятью [Amza 96].

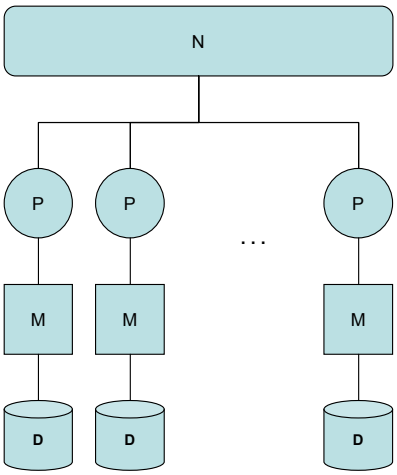
SD (*Shared-Disks*) - архитектура с разделяемыми дисками



- Каждый процессор имеет свою приватную оперативную память.
- Все диски доступны всем процессорам с одинаковым временем доступа.
- Межпроцессорные коммуникации осуществляются через высокоскоростную соединительную сеть.
- Аппаратная платформа: MPP или кластеры с разделяемыми дисками

SD архитектура представляет системы баз данных, в которых любой процессор имеет доступ к любому диску, однако каждый процессор имеет свою приватную оперативную память [Rahm 93]. Процессоры в таких системах соединены посредством некоторой высокоскоростной сети, позволяющей осуществлять передачу данных. Примерами параллельных систем баз данных с *SD* архитектурой являются IBM IMS [StricklandUW 82], Oracle Parallel Server [Linder 93] на nCUBE [Дубова 95] и VAXcluster [KronenbergLS 86], IBM Parallel Sysplex [King 97, Nick 97] и др.

SN (*Shared-Nothing*) - архитектура без совместного использования ресурсов



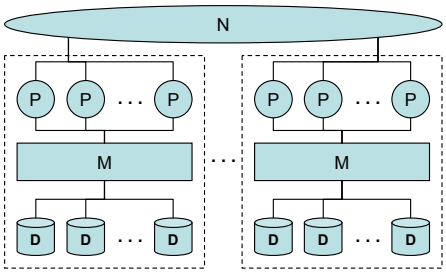
- Каждый процессор имеет свою приватную оперативную память и диск.
- Межпроцессорные коммуникации осуществляются через высокоскоростную соединительную сеть.
- Аппаратная платформа: MPP или кластеры с распределенными дисками

SN архитектура характеризуется наличием у каждого процессора собственной оперативной памяти и собственного диска. Как и в *SD* системах, процессорные узлы соединены некоторой высокоскоростной сетью, позволяющей организовывать обмен сообщениями между процессорами [DeWitt 92]. К настоящему моменту создано большое количество исследовательских прототипов и несколько коммерческих систем с *SN* архитектурой, использующих фрагментный параллелизм. В качестве примеров исследовательских прототипов *SN* систем можно привести следующие системы: ARBRE [Lorie 89], BUBBA [Boral 90], EDS [Skelton 92], GAMMA [DeWitt 90], KARDAMOM [Bultzingsloewen 88], PRISMA [Apers 92]. Примерами коммерческих систем с *SN* архитектурой являются NonStop SQL [Хаманн 97, Chambers 93, EnglertGH 95], Informix PDQ [Clay 93], NCR (Teradata) DBC/1012 [ЛисянскийС 97, Page 92], IBM DB2 PE [Baru 95] и др.

Виртуально-иерархическая классификация (расширение классификации Стоунбрейкера)

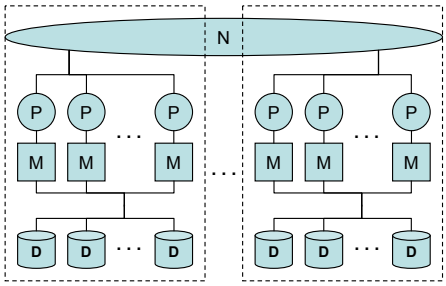
- *CE* (*Clustered-Everything*) – иерархическая архитектура с *SE* кластерами, объединенными по принципу *SN*.
- *CD* (*Clustered-Disk*) - иерархическая архитектура с *SD* кластерами, объединенными по принципу *SN*.
- *CDN* (*Clustered-Disk & Clustered-Nothing*) – иерархическая гибридная архитектура.
- …

CE (*Clustered-Everything*)



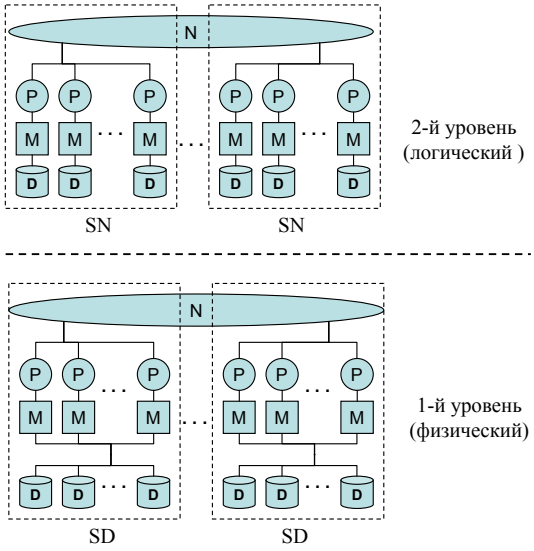
- *SE* кластеры объединены по принципу *SN*
- Межкластерныее коммуникации осуществляются через высокоскоростную соединительную сеть.
- Межпроцессорные коммуникации внутри *SE* кластера осуществляются через общую оперативную память.

CD (*Clustered-Disk*)



- *SD* кластеры объединены по принципу *SN*
- Граница *SD* кластеров распространены на общую (глобальную) соединительную сеть, так как в них может присутствовать собственная (локальная) соединительная сеть.
- Межпроцессорные коммуникации осуществляются через высокоскоростную соединительную сеть.

Гибридная архитектура C_D^N



CDN архитектура строится как набор однотипных *SD* кластеров, объединенных по принципу *SN*. Отличительной особенностью данной системной архитектуры является то, что на верхних уровнях системной иерархии *SD* кластеры рассматриваются как *SN* системы. Это выражается в том, что каждому процессорному узлу *логически* назначается отдельный диск.

5.2. Требования к параллельной системе баз данных

- высокая масштабируемость
- высокая производительность
- высокая доступность данных

Масштабируемость

- ускорение
- расширяемость

Ускорение

$\mathfrak{A}$ – множество различных конфигураций многопроцессорной системы

$\mathfrak{T}$ – набор тестов на производительность

$A,B \in \mathfrak{A}$

$Q \in \mathfrak{T}$

Коэффициент ускорения при переходе от A к B

$$a_{AB} = \frac{d_A t_{QA}}{d_B t_{QB}}$$

d_A - количество процессоров конфигурации A

d_B - количество процессоров конфигурации B;

t_{QA} - время, затраченное конфигурацией A на выполнение теста Q

t_{QB} - время, затраченное конфигурацией B на выполнение теста Q

Система демонстрирует *линейное ускорение*, если $a_{BC}=1$ для любых конфигураций B и C системы.

Расширяемость

$A_1, A_2, \ldots, A_n \in \mathfrak{A}$
$Q_1, Q_2, \ldots, Q_n \in \mathfrak{C}$
$|A_j|/|A_i| = |Q_j|/|Q_i|$, для всех $i,j = 1,\ldots,n$

Коэффициент расширяемости при переходе от A_k к A_m

$$e_{km} = \frac{t_{Q_k A_k}}{t_{Q_m A_m}}$$

Система демонстрирует *линейную расширяемость*, если $e_{km}=1$ для любых конфигураций системы.

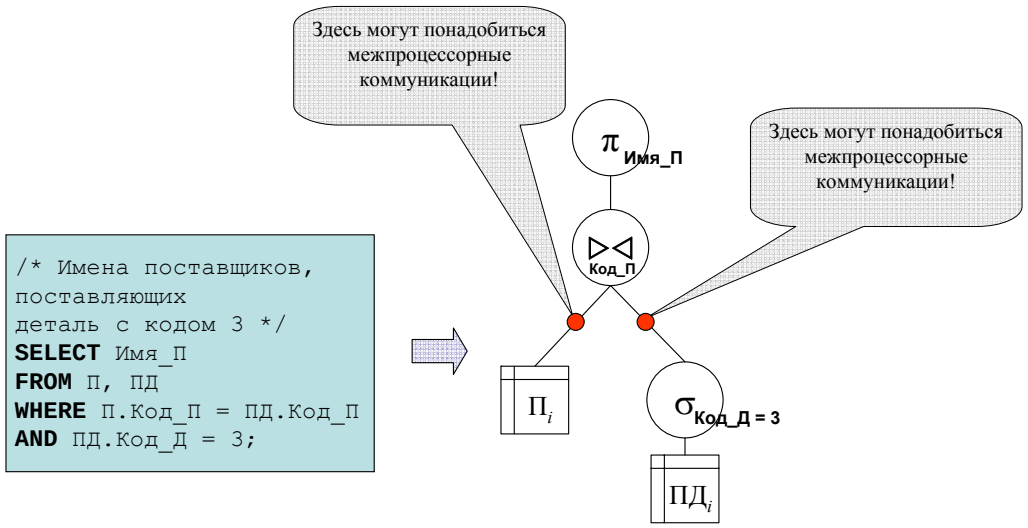
Высокая масштабируемость

Параллельная система *хорошо масштабируема*, если она демонстрирует ускорение и расширяемость, близкие к линейным

Производительность

- Межпроцессорные коммуникации
- Когерентность КЭШей
- Организация блокировок
- Балансировка загрузки

Межпроцессорные коммуникации



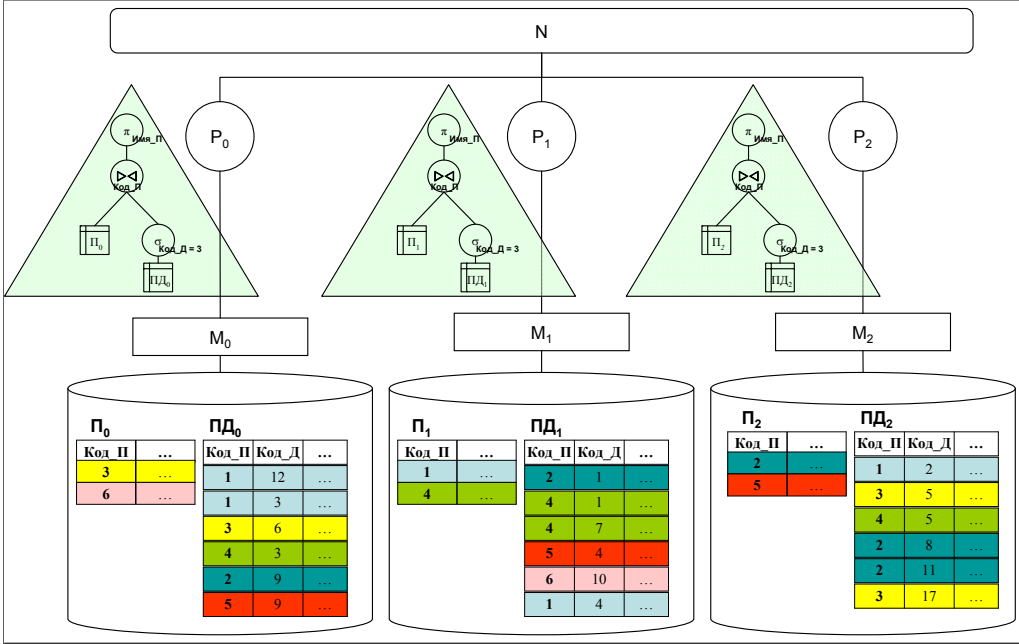
Фрагментный параллелизм требует межпроцессорных коммуникаций. Для их организации используется специальный оператор обмена ε. Для каждого входящего кортежа x оператор ε вычисляет функцию распределения **φ(*x*)** и пересылает кортеж на узел, номер которого выдала функция **φ**.

Пример, когда при выполнении запроса необходимы межпроцессорные коммуникации

n=3

$\varphi_{П}(x) = x.Код\_П \bmod 3$

$\varphi_{ПД}(x) = x.Код\_Д \bmod 3$



П0

Код_П	…
3	…
6	…

ПД0

Код_П	Код_Д	…
1	12	…
1	3	…
3	6	…
4	3	…
2	9	…
5	9	…

П1

Код_П	…
1	…
4	…

ПД1

Код_П	Код_Д	…
2	1	…
4	1	…
4	7	…
5	4	…
6	10	…
1	4	…

П2

Код_П	…
2	…
5	…

ПД2

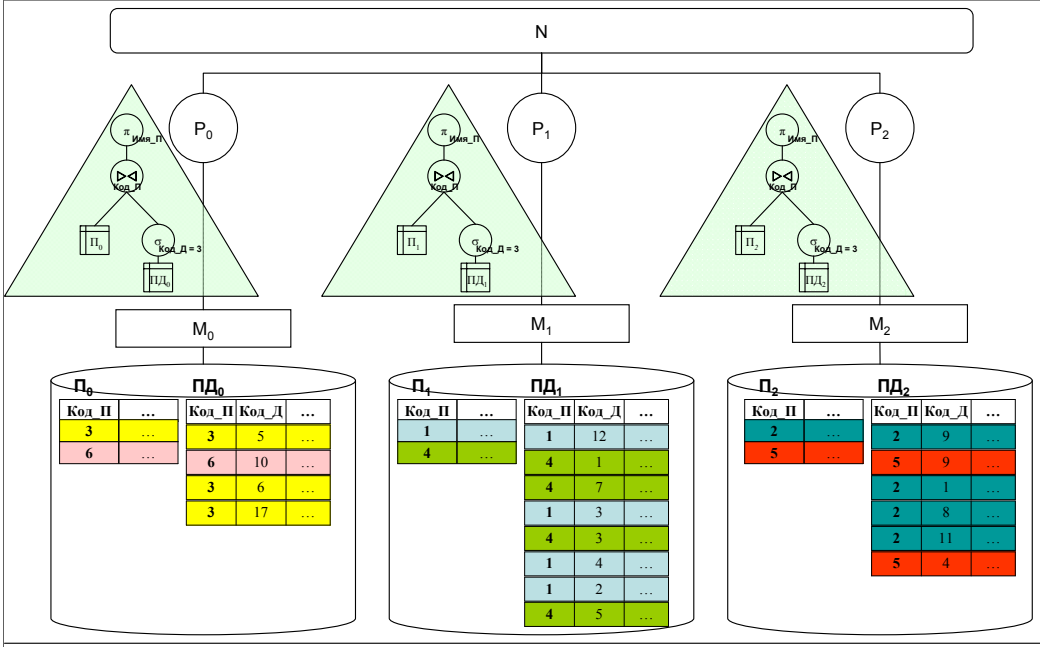
Код_П	Код_Д	…
1	2	…
3	5	…
4	5	…
2	8	…
2	11	…
3	17	…

Пример, когда межпроцессорные коммуникации при выполнении запроса не нужны

n=3

$\varphi_{П}(x) = x.Код\_П \bmod 3$

$\varphi_{ПД}(x) = x.Код\_П \bmod 3$



П0

Код_П	…
3	…
6	…

ПД0

Код_П	Код_Д	…
3	5	…
6	10	…
3	6	…
3	17	…

П1

Код_П	…
1	…
4	…

ПД1

Код_П	Код_Д	…
1	12	…
4	1	…
4	7	…
1	3	…
4	3	…
1	4	…
1	2	…
4	5	…

П2

Код_П	…
2	…
5	…

ПД2

Код_П	Код_Д	…
2	9	…
5	9	…
2	1	…
2	8	…
2	11	…
5	4	…

Балансировка загрузки

- *Перекосы данных*
- *Перекосы выполнения*

Перекос данных

- *На одном узле находится больше обрабатываемых данных, чем на другом*

Перекосы выполнения

- *Один узел выполняет более трудоемкую операцию, чем второй*

Доступность данных

$$коэффициент\_доступности\_БД = \frac{реальное\_время\_доступности\_БД}{требуемое\_время\_доступности\_БД}$$

Коэффициент доступности базы данных нестрого может быть определен как отношение межу промежутком времени, в течение которого база данных была действительно доступна пользователям, и промежутком времени, в течение которого пользователи требовали доступ к базе данных. Например, если пользователи требовали доступ к базе данных в течение 8 часов в день, а реально база данных была доступна только в течение 6 часов, то коэффициент доступности составляет 6/8 = 0.75 в течение 8-часового периода. Систему баз данных с *высокой доступностью данных* можно определить как систему, обеспечивающую прием запросов пользователей 24 часа в сутки с коэффициентом доступности не менее 0.99.

Высокая доступность данных

1) аппаратная отказоустойчивость
2) восстановление целостности базы данных после сбоя
3) оперативное восстановление базы данных
4) прозрачность для пользователя процессов восстановления системы

Аппаратная отказоустойчивость является основным фактором обеспечения высокой доступности данных в параллельных системах баз данных с большим количеством процессорных узлов. Под аппаратной отказоустойчивостью понимают сохранение общей работоспособности системы при одиночном отказе таких аппаратных компонент, как процессор, модуль памяти, диск, и каналов доступа к перечисленным компонентам. В частности, одиночный отказ любого устройства не должен привести к потере целостности базы данных и тем более к физической утрате какой-то части базы данных.

Восстановление целостности базы данных после сбоя предполагает поддержку ACID транзакций и журнализацию изменений. Данные возможности поддерживаются большинством современных СУБД с архитектурой клиент-сервер.

Оперативное восстановление базы данных предполагает восстановление нормальной работоспособности системы с сохранением режима обслуживания пользователей. При этом коэффициент доступности данных может временно уменьшаться.

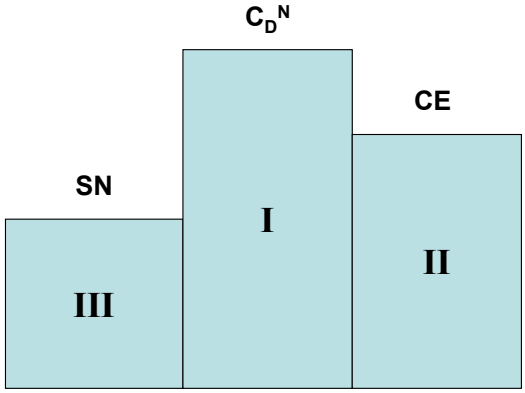
Прозрачность для пользователя процессов восстановления системы предполагает незначительное уменьшение коэффициента доступности базы данных во время сбоя и последующего восстановления. Сложность данной проблемы заключается в том, что выход из строя одного из дисков может привести к серьезному дисбалансу загрузки процессоров, например, в силу удвоения нагрузки на узел, содержащий копию утраченных данных. Возможное решение указанной проблемы заключается в том, чтобы фрагментировать копию диска по другим дискам таким образом, чтобы она допускала параллельный доступ.

5.3. Сравнительный анализ архитектур

Критерий	***Архитектура***					
	SE	***SD***	***SN***	***CE***	***CD***	C_D^N
Масштабируемость	0	1	2	3	3	3
Доступность данных	0	1	3	1	2	2
Баланс загрузки	3	2	0	2	1	1
Межпроцессорные коммуникации	3	0	0	2	1	1
Когерентность кэшей	2	0	3	2	0	3
Организация блокировок	2	0	3	2	0	3
Сумма баллов	10	4	11	12	7	13

0 - "плохо“
1 - "удовлетворительно“
2 - "хорошо"
3 - "превосходно"

Наиболее перспективные архитектуры

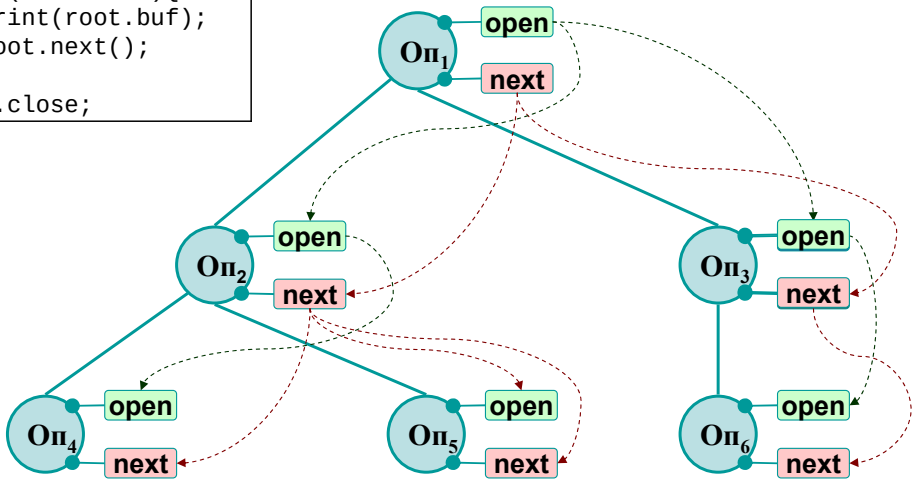


Параллельные системы баз данных

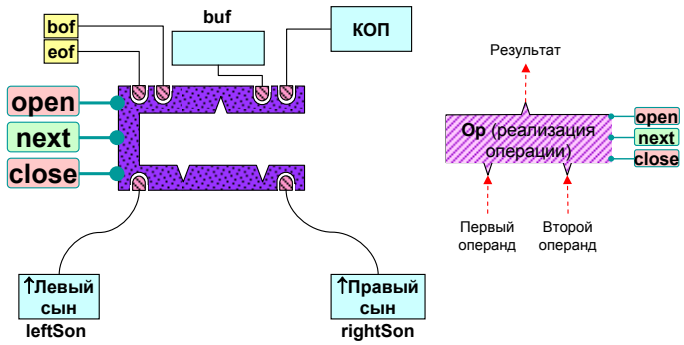
6. Обработка запросов

Итераторная модель

```
root.open();
root.next();
while(!root.eof){
   print(root.buf);
   root.next();
};
root.close;
```



Скобочный шаблон (node)



Для унифицированного представления узлов дерева запроса используется класс «скобочный шаблон». В качестве основных методов здесь фигурируют **open**, **close** и **next**, реализующие итератор. Основными атрибутами скобочного шаблона являются

- выходной буфер **buf**, в который помещается очередной кортеж результата;
- КОП: код реляционной операции, реализуемой данным узлом;
- указатель на скобочный шаблон левого сына;
- указатель на скобочный шаблон правого сына («пусто» для унарных операций).

Сам по себе скобочный шаблон не содержит конкретной реализации реляционной операции. Однако, после оптимизации запроса СУБД «вставляет» в каждый скобочный шаблон ту или иную реализацию соответствующей реляционной операции. Например, для операции соединения мы можем выбирать «соединение вложенными циклами», «соединение слиянием», соединение хешированием» и др. Связь скобочного шаблона с конкретной реализацией операции осуществляется путем добавления еще одного специального атрибута в скобочный шаблон – «указатель на функцию реализации операции». В качестве параметра данной функции должен передаваться указатель на объемлющий скобочный шаблон.

Схема реализации методов *reset* и *next*

```
void node.open{
   node.bof = 1;
   node.eof = 0;
   node.op.open(node);
};
```

```
buffer node.next{
   node.bof = 0;
   node.op.next(node);
   if(node.buf==EOF) node.eof = 1;
};
```

Реализация соединения в оперативной памяти NLMJ (Nested Loops Memory Join)

- $R \underset{(A)}{\triangleright\triangleleft} S$
- R – опорное отношение целиком помещается в оперативную память
- S – тестируемое отношение
- Атрибуты:
  - M – массив в оперативной памяти (для R)
  - m – указатель на текущий элемент в M

NLMJ: схема реализации метода open

```
void* open(node){
   node.leftSon.open();
   m = 0;
   do{/* Заполняем массив M кортежами опорного (левого) отношения */
     node.leftSon.next();
     M(m) = node.leftSon.buf;
     m++;
   } while (!node.leftSon.eof);
   m = 0;
   node.rightSon.open();
   node.rightSon.next();
};
```

NLMJ: схема реализации метода next

```
void* next(node){
   while(!node.rightSon.eof){// Сканируем тестируемое (правое) отношение
      while(M(m)!=EOF){  // Сканируем опорное (левое) отношение
         if(MATCH_A(M(m),node.rightSon.buf)){
            node.buf=JOIN_A(M(m),node.rightSon.buf);
            m++;
            return;
         };
         m++;
      };
      m=0;
      node.rightSon.next();
   };
   node.buf=EOF;
};
```

Проверяет на совпадение значений атрибута A у двух кортежей

Выполняет естественное соединение двух кортежей по атрибуту A

Реализация соединения вложенными циклами NLDJ (Nested Loops Disk Join)

- $R \underset{(A)}{\triangleright\!\triangleleft} S$
- R – опорное (меньшее) отношение
- S – тестируемое (большее) отношение

NLDJ: схема реализации метода open

```
void* open(node){
   node.leftSon.open();
   node.rightSon.open();
   node.leftSon.next();
   node.rightSon.next();
};
```

NLDJ: схема реализации метода next

```
void* next(node){
   while(!node.rightSon.eof){// Сканируем тестируемое (правое) отношение
      while(!node.leftSon.eof){// Сканируем опорное (левое) отношение
         if(MATCH_A(node.leftSon.buf,node.rightSon.buf)){
            node.buf=JOIN_A(node.leftSon.buf,node.rightSon.buf);
            node.leftSon.next();
            return;
         };
         node.leftSon.next();
      };
      node.leftSon.close();
      node.leftSon.open();
      node.rightSon.next();
   };
   node.buf=EOF;
};
```

Проверяет на совпадение значений атрибута A у двух кортежей

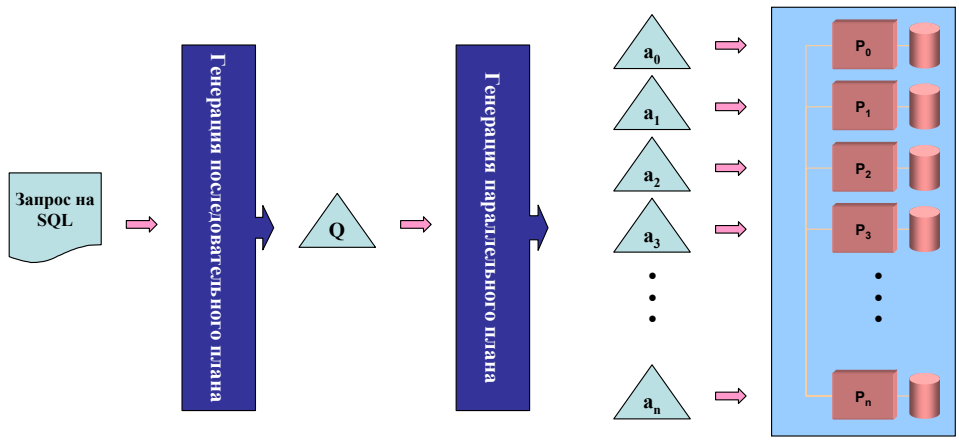
Выполняет естественное соединение двух кортежей по атрибуту A

Можно заменить на `node.leftSon.reset();`

Параллельные системы баз данных

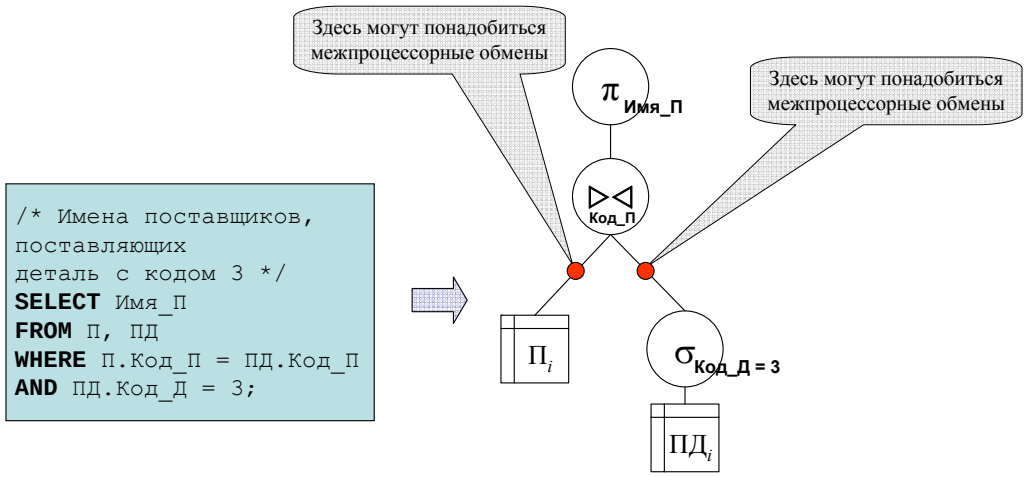
7. Организация межпроцессорных обменов

Общая схема обработки запроса



На данном слайде изображена общая схема обработки запроса в параллельной СУБД. В соответствие с этой схемой запрос на языке SQL преобразуется в некоторый *последовательный план*. Данный последовательный план преобразуется в *параллельный план,* представляющий собой совокупность *n* идентичных параллельных *агентов*. Здесь *n* обозначает количество процессорных узлов.

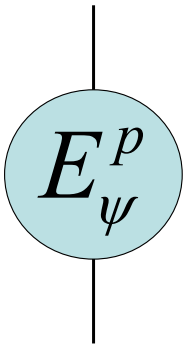
Межпроцессорные обмены




```
/* Имена поставщиков,
поставляющих
деталь с кодом 3 */
SELECT Имя_П
FROM П, ПД
WHERE П.Код_П = ПД.Код_П
AND ПД.Код_Д = 3;
```


Фрагментный параллелизм требует межпроцессорных коммуникаций.

Оператор exchange



p: порт обмена

ψ: функция распределения

Для организации межпроцессорных обменов используется специальный оператор **exchange**. Оператор **exchange** реализуется на основе использования стандартного скобочного шаблона и может быть добавлен в качестве узла в любое место дерева запроса. Оператор **exchange** имеет два специальных параметра, определяемых пользователем:

p: порт обмена – целое положительное число, задающее уникальный номер оператора обмена в плане запроса;

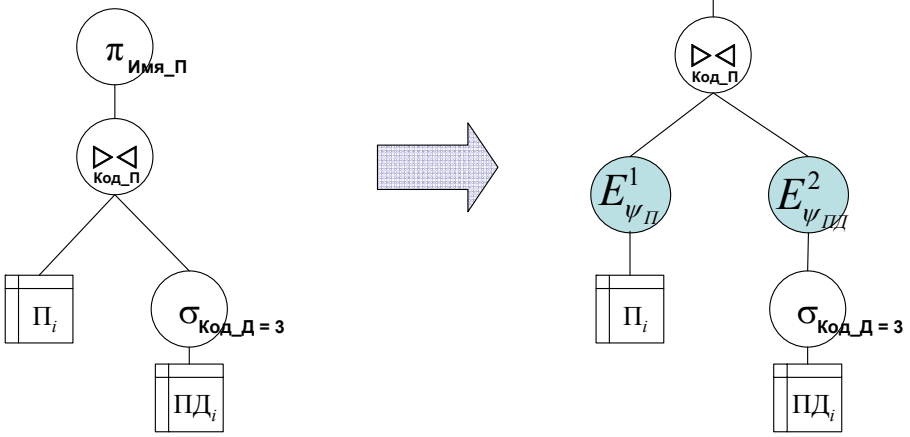
ψ: функция распределения для произвольного входного кортежа *x* определяет номер процессорного узла, на котором указанный кортеж должен обрабатываться.

Схема работы оператора E_{ψ}^{p}

k = номер моего узла;
p = номер моего порта;
x = входной кортеж;
if $\psi(x) = k$ **then**
    поместить *x* в свой выходной буфер;
else
    переслать *x* на процессорный узел с номером $\psi(x)$ и поместить его в выходной буфер оператора exchange с портом номер *p*;
end if

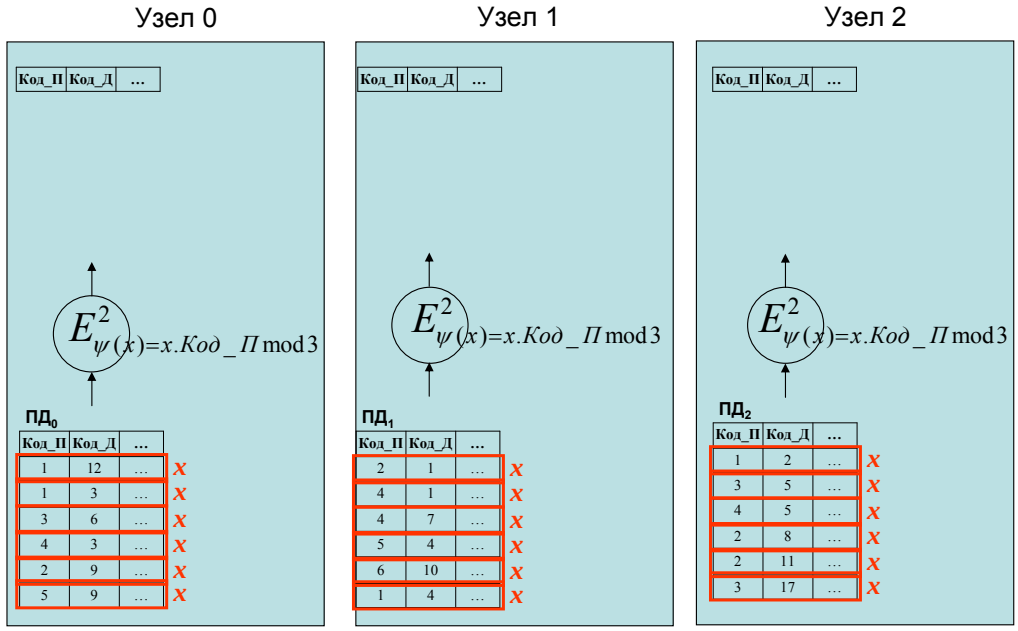
Создание параллельного агента

$$\psi_{П}(x) = f_{Код\_П}(x.Код\_П)$$
$$\psi_{ПД}(y) = f_{Код\_П}(y.Код\_П)$$



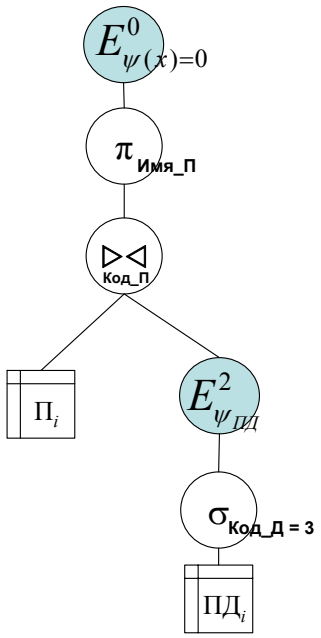
В общем случае, когда отношения П и ПД фрагментированы произвольным образом, нам необходимо перед соединением вставить два оператора exchange. Верхний оператор exchange обеспечивает слияние фрагментов результирующего отношения в общую результирующую таблицу на нулевом узле.

Пример работы оператора обмена



Левое отношение фрагментировано по атрибуту соединения

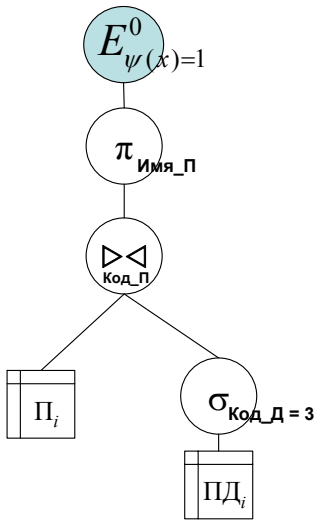
$$\varphi_{П}(x) = f_{Код\_П}(x.Код\_П)$$
$$\psi_{ПД}(y) = f_{Код\_П}(y.Код\_П)$$



Здесь мы предполагаем, что отношение *П* фрагментировано по атрибуту соединения с помощью некоторой функции $\varphi_П$, а отношение *ПД* фрагментировано по некоторому другому атрибуту, не являющемуся атрибутом соединения. В данном контексте необходимо вставить в дерево запроса один оператор обмена **exchange**. В качестве функции распределения оператора **exchange** указывается функция $\psi_{ПД}$, а в качестве номера порта обмена - любой свободный на данный момент номер.

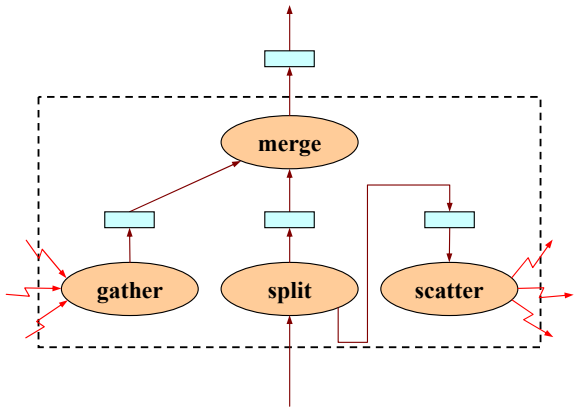
Оба отношения фрагментированы по атрибуту соединения

$$\varphi_{П}(x) = f_{Код\_П}(x.Код\_П)$$
$$\varphi_{ПД}(y) = f_{Код\_П}(y.Код\_П)$$



В том случае, когда оба отношения фрагментированы по атрибуту соединения с помощью общей функции фрагментации *f*, необходимость в межпроцессорных коммуникациях перед выполнением соединения отсутствует.

Структура оператора exchange



Оператор **exchange** является составным оператором и включает в себя четыре оператора: **gather**, **scatter**, **split** и **merge**. Все указанные операторы реализуются на базе стандартного скобочного шаблона.

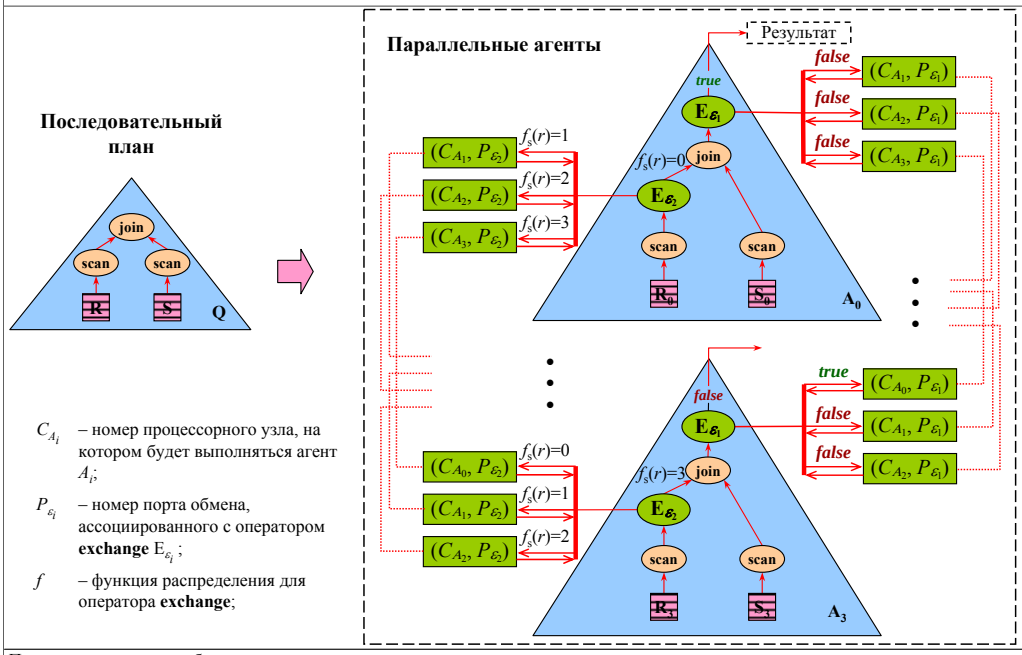
Оператор **split** - это бинарный оператор, который осуществляет разбиение кортежей, поступающих из входного потока (ассоциируется с левым сыном), на две группы: *свои* и *чужие*. Свои кортежи - это кортежи, которые должны быть обработаны на данном процессорном узле. Эти кортежи направляются в выходной буфер оператора **split**. Чужие кортежи, то есть кортежи, которые должны быть обработаны на процессорных узлах, отличных от данного, помещаются оператором **split** в выходной буфер правого сына, в качестве которого фигурирует оператор **scatter**. Здесь выходной буфер оператора **split** играет роль входного потока данных.

Нульарный оператор **scatter** извлекает кортежи из своего выходного буфера и пересылает их на соответствующие процессорные узлы, используя заданный номер порта. Запись кортежа в порт может быть завершена только после того, как реципиент выполнит операцию чтения из данного порта.

Нульарный оператор **gather** выполняет перманентное чтение кортежей из указанного порта со всех процессорных узлов, отличных от данного. Считанные кортежи помещаются в выходной буфер оператора **gather**.

Оператор **merge** определяется как бинарный оператор, который забирает кортежи из выходных буферов своих сыновей и помещает их в собственный выходной буфер.

Пример преобразования последовательного плана в параллельный



Преобразование последовательного плана в параллельный достигается путем вставки оператора обмена **exchange** в соответствующие места дерева плана запроса. На завершающем этапе агенты рассылаются на соответствующие процессорные узлы, где *интерпретируются* исполнителем запросов. Результаты выполнения агентов объединяются *корневым* оператором **exchange** на нулевом процессорном модуле.

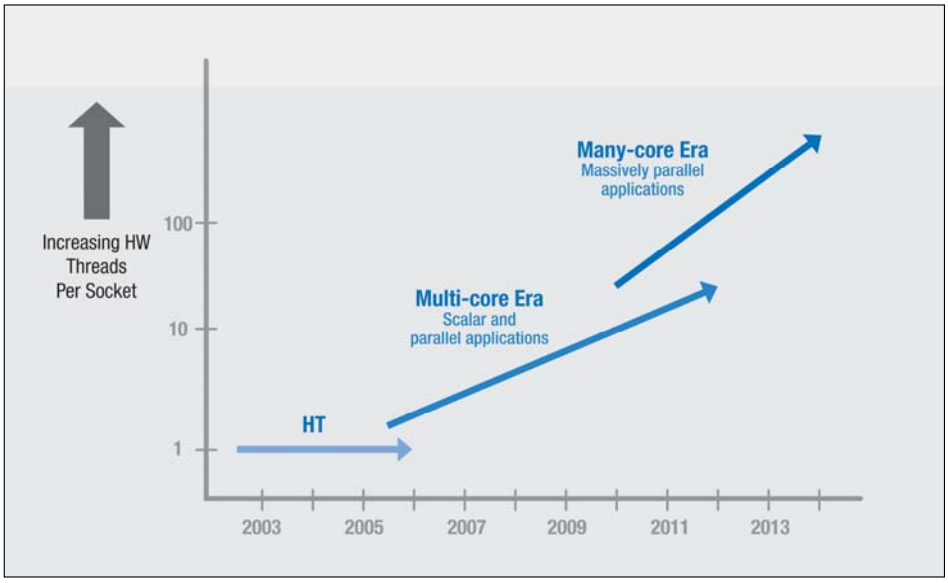
Параллельные системы баз данных

8. Балансировка загрузки в многопроцессорных иерархиях

Предпосылки формирования многопроцессорных иерархий

- Многоядерные процессоры
- Кластеры
- Grid

Многоядерные процессоры

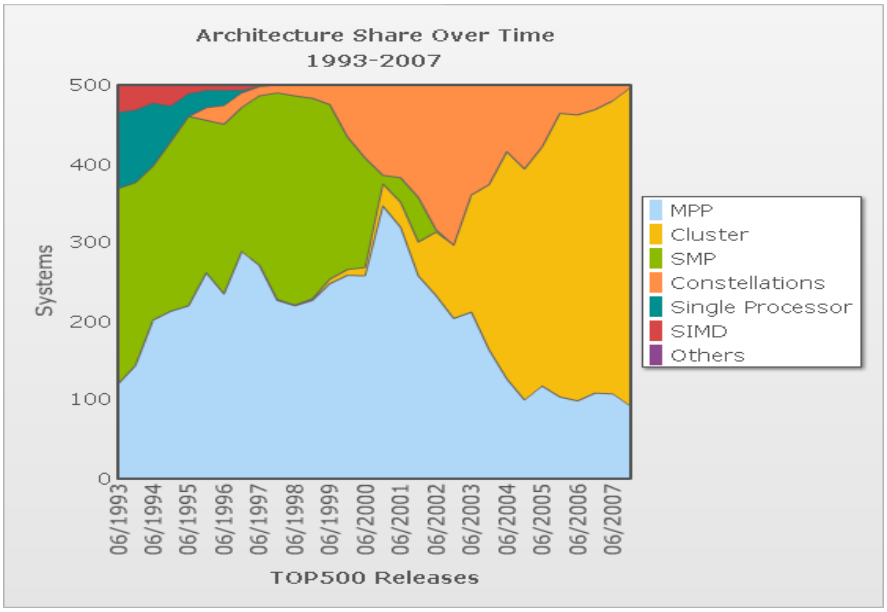


Platform 2015: Intel Processor and Platform Evolution for the Next Decade. White Paper. -Intel Corporation, 2005.

Мультиядерность: 2-8 ядер и до 100 нитей

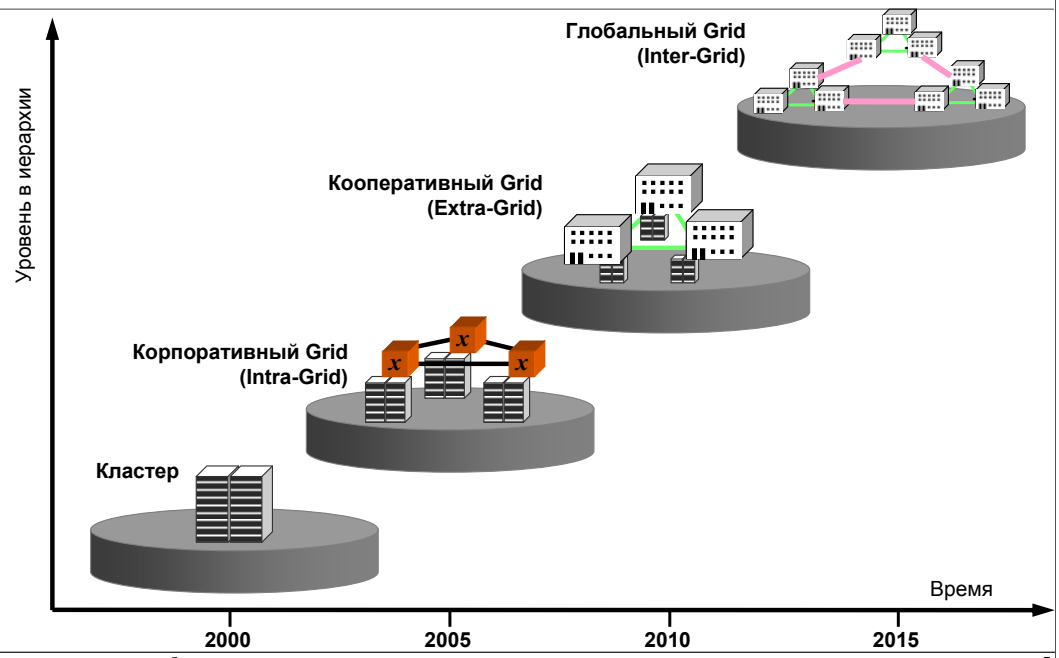
Многоядерность: 10-100 ядер и до 1000 нитей

Кластеры

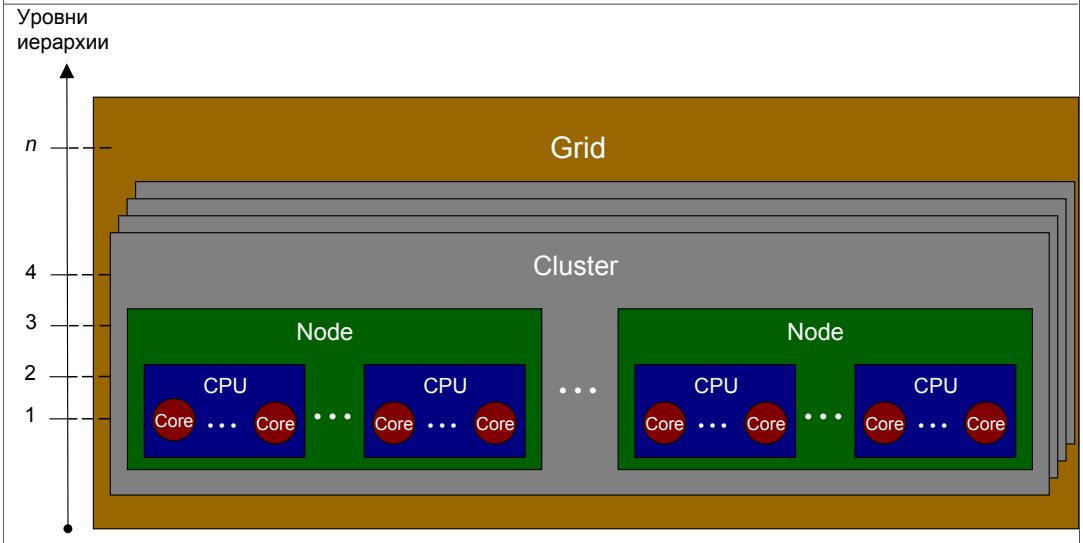


http://www.top500.org/overtime/list/30/archtype

Grid


Глобальный Grid
(Inter-Grid)
Кооперативный Grid
(Extra-Grid)
Корпоративный Grid
(Intra-Grid)
Кластер
Уровень в иерархии
Время
2000
2005
2010
2015

Многопроцессорная иерархия



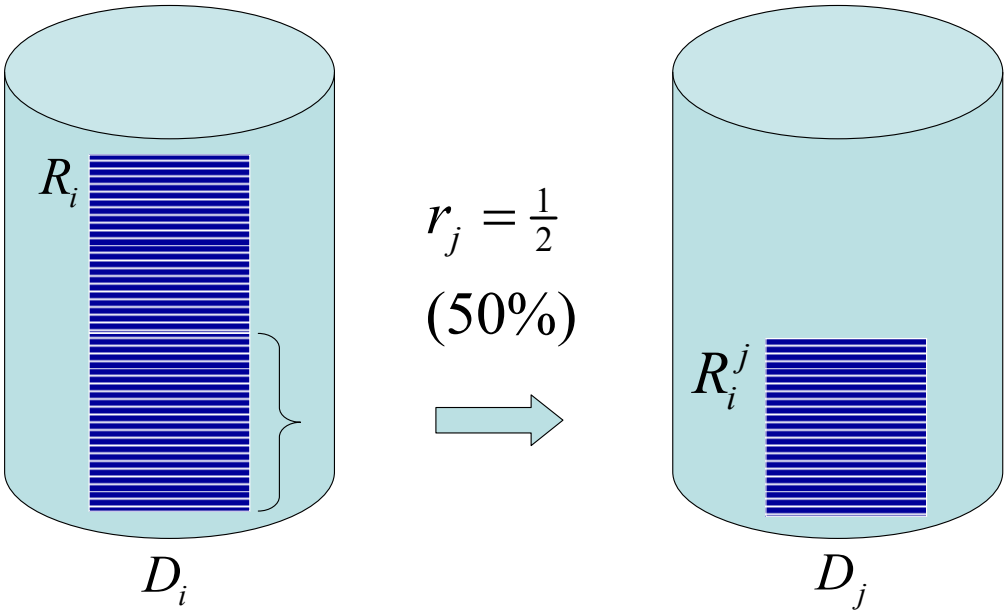
Распределение данных

- **Отношение (таблица) разбивается на фрагменты, располагающиеся на различных дисках**
- **Фрагмент делится на сегменты, между которыми определено отношение порядка**
- **С каждым фрагментом связывается константа *c*, определяющая дину его сегментов в кортежах (записях):**
  - **все сегменты, кроме последнего, имеют длину *c;***
  - **последний сегмент имеет длину ≤c**
- **Сегмент является наименьшей единицей репликации**

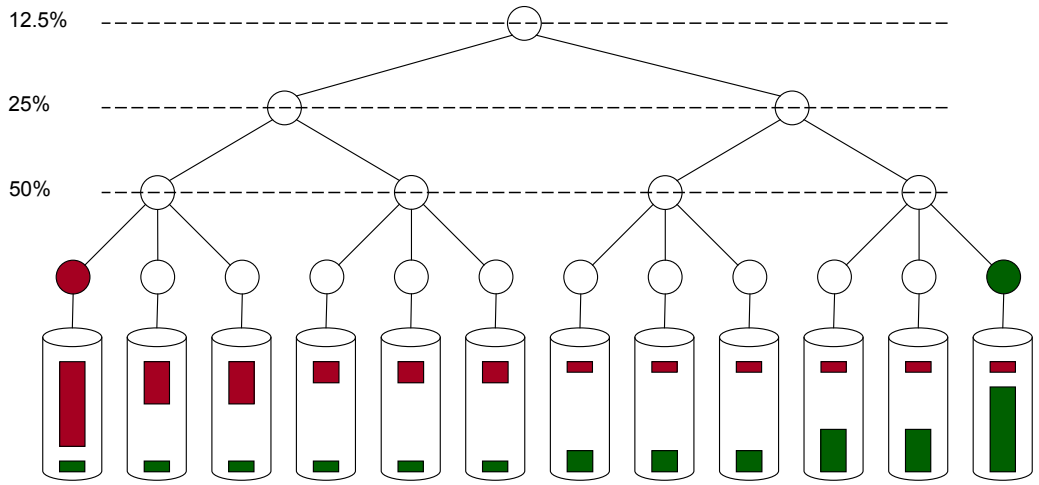
Зеркалирование

- **Фрагмент может иметь несколько (возможно неполных) зеркальных копий, называемых репликами, которые располагаются на других узлах**
- **На каждом узле может находиться не более одной реплики данного фрагмента**
- **Содержимое реплики однозначно определяется *коэффициентом зеркалирования r*, назначенным узлу, на котором хранится реплика**

Построение реплики



Зеркалирование в многопроцессорной иерархии



Фрагментация и репликация

$\varphi_X : X \to N$ – *функция фрагментации* отношения X (для любого $x \in X$ значение $\varphi_X(x)$ определяет номер узла, на котором хранится кортеж *x*).

X_k – *фрагмент отношения* X, хранящийся на *k*-том узле: $X_k = \{x \mid x \in X, \varphi_X(x) = k\}$.

X_k^m – *реплика фрагмента* X_k, хранящаяся на *m*-том узле. В частности, имеем $X_k^k = X_k$.

Функция распределения

$\psi_X : X \to N$ – *функция распределения* отношения X (для любого $x \in X$ значение $\psi_X(x)$ определяет номер узла, на котором кортеж x должен обрабатываться).

Балансировка естественного соединения

Рассмотрим естественное соединение двух отношений R и S по общему атрибуту A. Отношение R играет роль опорного, S – тестируемого, то есть $T(R) < T(S)$.

Соединение хешированием в основной памяти

MHJ (Memory Hash Join):

1. Строит в оперативной памяти хеш-таблицу для опорного отношения (первый операнд). При этом предполагается, что опорное отношение (его фрагмент) полностью помещается в оперативную память.
2. Выполняет соединение, просматривая за один проход все кортежи тестируемого отношения (второй операнд).

Фрагментация и репликация

- Справа всегда вставляется exchange
- m-тый агент завершил обработку S_m^m;
  k-тый агенту осталось обработать v сегментов S_k^k.
- m-тому агенту на правый вход подается w конечных сегментов S_m^k, где $w = \min(T(S_k^m), v/2)$.
- k-тому агенту на правый вход подается $v - w$ начальных сегментов необработанной части конечных сегментов S_k^k.