Распределенные системы: отказоустойчивость

Характерной чертой распределенных систем является возможность частичного отказа — сбоя в одном из компонентов распределенной системы.

При создании распределенной системы важно, чтобы она могла автоматически восстанавливаться после частичных отказов.

Распределенная система в процессе восстановления должна продолжать работать приемлемым образом, сохраняя в случае отказов определенную степень функциональности.

Современные информационные системы предъявляют высокие требования к программно-аппаратному обеспечению, и, в частности, к надежности и катастрофоустойчивости.

В связи с этим актуальным является проблема создания информационной системы с высокими показателями надежности и катастрофоустойчивости.

Есть два аспекта надежности вычислительных систем:
- надежность хранения данных;
- непрерывность предоставления сервиса.

Очевидно, что создать абсолютно надежную систему технически невозможно, однако для каждой системы принимаются необходимые меры для повышения надежности.

Естественно, затраты должны быть адекватны возможным потерям, связанным с отказом системы в целом или какой-либо ее части, и, вследствие этого, с прекращением предоставления сервиса (незапланированный простой).

Также необходимо учитывать и плановые простои, связанные с регламентными работами по обслуживанию системы.

Катастрофоустойчивость предполагает в первую очередь обеспечение сохранности данных, а также возможность восстановить работу после крупной локальной аварии или глобального катаклизма, причем теми же средствами обеспечивается заодно и должная степень надежности (традиционная, «локальная», отказоустойчивость) всех или критически важных подсистем.

Поскольку компоненты распределены, то в случае массовых отказов на одной площадке основную работу можно перенести на другую площадку.

Отказоустойчивость

Отказоустойчивость напрямую связана с надежностью систем. Надежность охватывает множество требований к распределенным системам, из которых важнейшими являются:

- доступность(availability);
- безотказность(reliability);
- безопасность(safety);
- ремонтопригодность(maintainability).

Доступность (готовность)

Свойство системы находиться в состоянии готовности к работе.
Обычно доступность – это вероятность того, что система в данный момент времени будет правильно работать и окажется в состоянии выполнить свои функции, если пользователи того потребуют.

Система с высокой степенью доступности – это такая система, которая в произвольный момент времени, скорее всего, находится в работоспособном состоянии.

Классификация систем по показателю доступности (готовности)

Класс готовности	Показатель готовности, %	Продолжительность простоев за год	Степень надежности
1	90	более 1 месяца	надежная система
2	99	менее 4 дней	надежная система
3	99.9	менее 9 часов	высоконадежная система
4	99.99	около 1 часа	отказоустойчивая система
5	99.999	около 5 минут	безотказная система
6	99.9999	около 30 секунд	безотказная система

Безотказность

Свойство системы работать без отказов в течение продолжительного времени. Определяется в понятиях временного интервала, а не момента времени.

Примеры:

- система отказывает на 1 мс за 1 час → доступность 99.9999%, но низкая безотказность;
- система никогда не отказывает, но выключается раз в год на 2 недели → высокая безотказность, при доступности 96%

Безопасность

Безопасность определяет, насколько катастрофична ситуация временной неспособности системы должным образом выполнять свою работу.

По-настоящему безопасную систему построить крайне тяжело.

Ремонтопригодность

Ремонтопригодность определяет, насколько сложно исправить неполадки в описываемой системе.

В настоящее время считается, что наиболее оптимальной схемой построения отказоустойчивой и катастрофоустойчивой информационной системы является **кластеризация**, т.е. *создание многомашинных комплексов с разделяемой массовой памятью* и возможностью переключения приложений между членами кластера в случае их отказов (failover).

Поскольку такой кластер может строиться из (недорогих) машин общего назначения, и при этом достигается достаточно высокая отказоустойчивость (непрерывность предоставления сервиса), с точки зрения затрат эта схема считается оптимальной.

Основные понятия

Система *отказывает* (fail), если она не в состоянии выполнять свою работу, соответственно *ошибкой* (error) называют такое состояние системы, которое может привести к ее неработоспособности. Причиной ошибки является *отказ* (fault).

Построение надежных систем тесно связано с управлением отказами – здесь имеется в в виду нечто среднее между предотвращением, исправлением и предсказанием отказов.

Соответственно для этих целей наиболее важной частью является отказоустойчивость. Под *отказоустойчивостью* (fault tolerance) понимается способность системы предоставлять услуги даже при наличии отказов.

Отказы

• Проходные (transient faults) – происходят однократно и больше не повторяются, даже если удается повторить ситуацию.

• Перемежающиеся (intermittent faults) – появляются и пропадают “когда захотят”, а потом появляются снова. Трудно диагностировать.

• Постоянные (permanent faults) – отказы, которые продолжают свое существование до тех пор, пока отказавший компонент не будет заменен.

Статистика причин отказов в системах обработки информации

- отказы дисков - 27%,
- отказы сервера или его ядра - 24%,
- отказы в программах - 22%,
- отказы в коммуникационном оборудовании - 11%,
- отказы в каналах передачи данных - 10%,
- отказы из-за ошибок персонала - 6%.

Модели отказов

В распределенной системе очень много зависимостей, поэтому отказ может нарушить работу не только одного узла, но и всей системы в целом.

Для классификации отказов разработано очень много различных схем. Одна из таких схем, применимых к распределенным системам:

Тип отказа	Описание
Поломка	Сервер перестал работать, хотя до момента отказа работал правильно. Важной особенностью является то, что после остановки сервера никаких признаков его работы не наблюдается.
Пропуск данных	Сервер неправильно реагирует на входящие запросы
Пропуск приема	Сервер неправильно принимает входящие запросы
Пропуск передачи	Сервер неправильно отправляет сообщения
Ошибка синхронизации	Реакция сервера происходит не в определенный момент времени
Ошибка отклика	Отклик сервера неверен
Ошибка значения	Сервер возвращает неправильное значение
Ошибка передачи состояния	Сервер отклоняется от верного потока управления. Возникает тогда, когда на сервере не предусмотрена защита от некорректно сформулированного запроса.
Произвольная ошибка *(византийские ошибки)*	Сервер отправляет случайные сообщения в случайные моменты времени

Маскирование ошибок при помощи избыточности

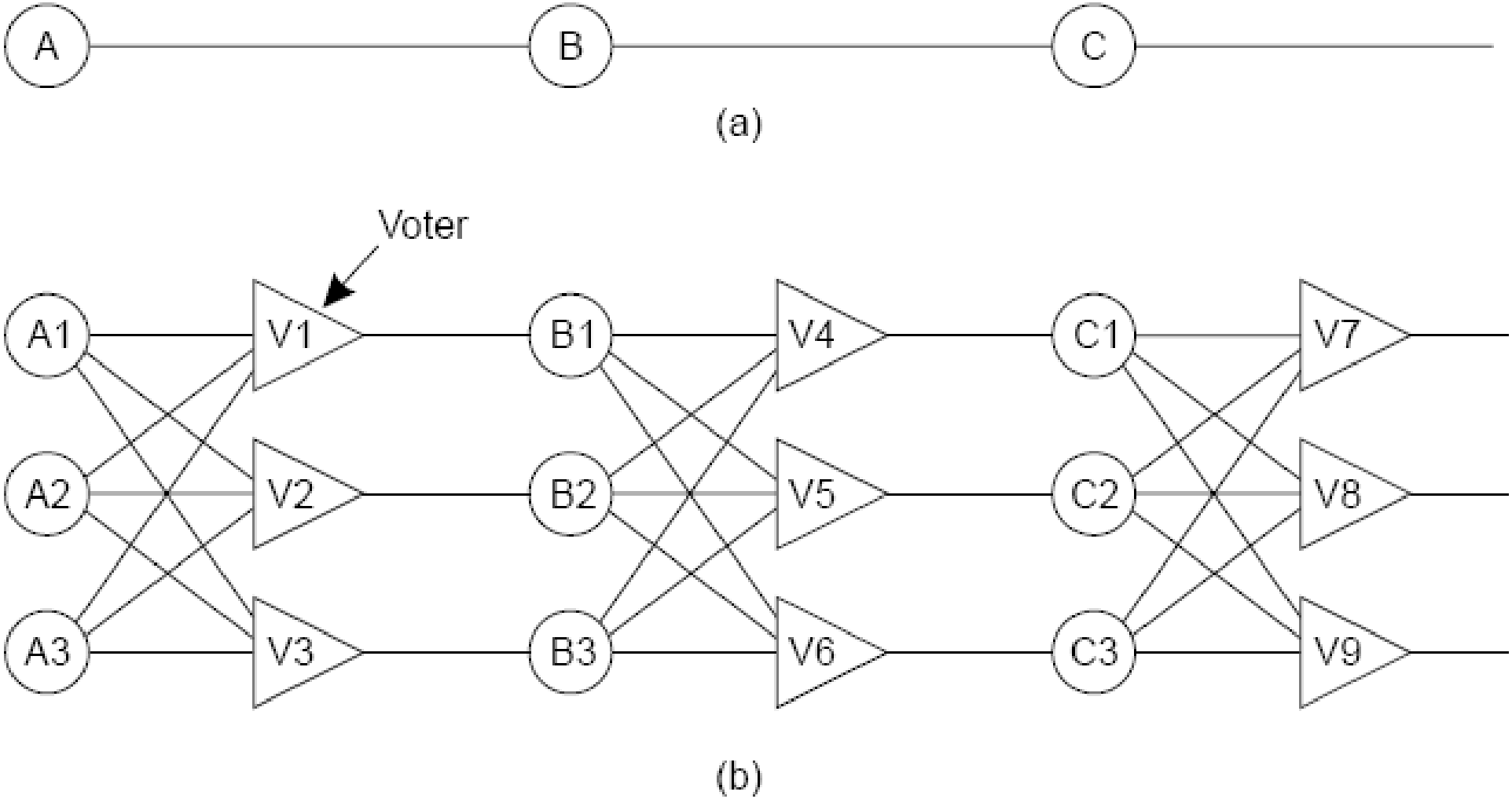
Отказоустойчивая система должна пытаться маскировать факты ошибок от других процессов.

Избыточность – это основной мед маскировки.

Возможно примение 3 типов избыточности:

- информационная – например добавление избыточной информации к сообщению, по которой можно определить или исправить сбойные биты;
- временная – при необходимости действие может быть выполнено еще раз;
- физическая – добавляется дополнительное оборудование или процессы (например тройное модульное резервирование).

Triple Modular Redundancy (TMR)



Отказоустойчивость процессов

Основной подход к защите от последствий отказа процессов — объединить несколько идентичных процессов в группу.

Основное свойство всех подобных групп состоит в том, что когда сообщение посылается группе, его получают все члены группы.

Т.о. если один из членов группы перестает работать, можно надеятся на то, что его место займет другой

Группы

• Однораноговые группы — симметричны и не имеют единой точки сбоя, однако сложный процесс принятия решений.

• Иерархические группы — единая точка сбоя и простой механизм принятия решений.

Членство в группе

Необходимы методы создания и уничтожения групп, добавления процессов в группу и удаление из нее.

Один из вариантов — создание сервера групп.

Другой вариант — распределенное управление членством. Например при использовании надежной групповой рассылки.

Проблемы групп

• есть возможность объявить о добровольном прекращении членства в группе, но нет возможности объявить об аварийном — такая ситуация определяется другими членами экспериментально.

• добавление/удаление члена группы должно происходить синхронно с обработкой сообщений с данными.

• что будет происходить при отказе сразу нескольких машин, в результате которого группа не сможет продолжать функционирование?

Соглашения в системах с ошибками

Рассмотрим два «простых» (идеализированных) случая систем с ошибками:

• проблема системы с идеальными процессами, линии связи между которыми могут терять данные — *проблема двух армий* (two-army problem).

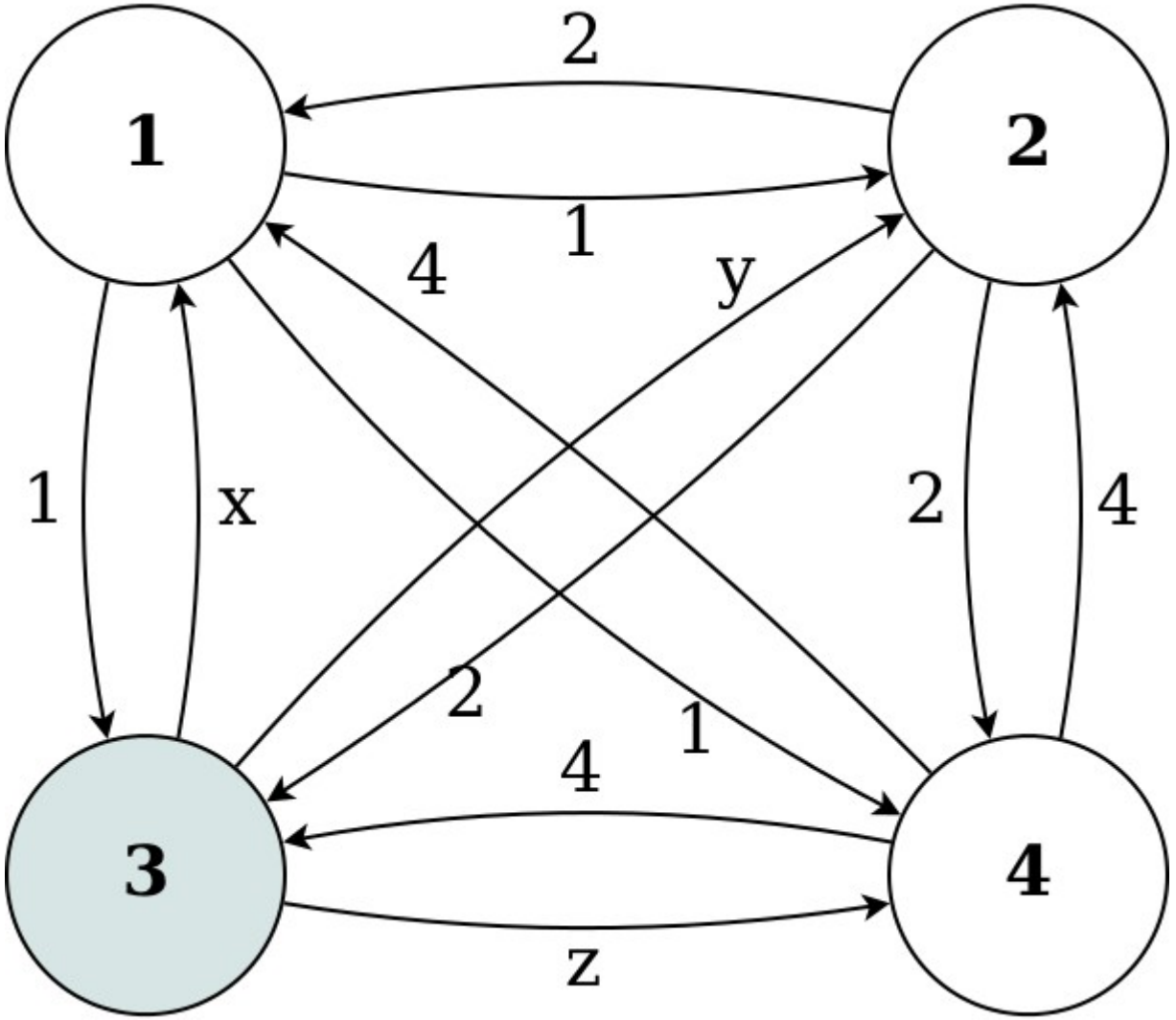
• проблема системы с идеальными линиями связи и ненадежными процессами — *проблема византийских генералов* (Byzantine generals problem).

Пример: рекурсивный алгоритм решения проблемы византийских генералов

1. Каждый генерал посылает каждому из прочих генералов сообщение о численности своей армии.
2. Результаты объявленные на шаге 1, собираются в виде векторов.
3. Каждый генерал отсылает свой вектор всем остальным генералам.
4. Каждый генерал проверяет полученные векторы. Если одно из значений находится в большинстве — оно помещается в итоговый вектор, если нет преимуществ — помечается, как UNKNOWN.

Доказано, что в системе с m дефектными процессами соглашение может быть достигнуто только при 2m+1 правильно функционирующих процессах.

Рекурсивный алгоритм решения проблемы византийских генералов: шаг 1



Рекурсивный алгоритм решения проблемы византийских генералов: шаги 2-4

Шаг 2

1 Получено: (1, 2, x, 4)
2 Получено: (1, 2, y, 4)
3 Получено: (1, 2, 3, 4)
4 Получено: (1, 2, z, 4)

Шаг 3

1 Получено:
2:(1, 2, y, 4)
3:(a, b, c, d)
4:(1, 2, z, 4)

2 Получено:
1:(1, 2, x, 4)
3:(e, f, g, h)
4:(1, 2, z, 4)

Шаг 3

3 Получено:
1:(1, 2, x, 4)
2:(1, 2, y, 4)
4:(1, 2, z, 4)

4 Получено:
1:(1, 2, x, 4)
2:(1, 2, y, 4)
3:(i, j, k, l)

Шаг 4: (1, 2, UNKNOWN, 4)

Ошибки связи

При взаимодействии точка-точка ошибки можно разделить на 5 классов:

• клиент не в состоянии обнаружить сервер;
• потеря сообщения с запросом от клиента к серверу;
• поломка сервера после получения запроса — различное поведение и проблемы (много проблем) при поломке:
- после обработки запроса;
- до обработки запроса.
• потеря ответного сообщения от сервера к клиенту;
• поломка клиента после получения ответа (процессы на сервере становятся сиротами.

Борьба с сиротами (orphans) при поломке клиента

• *истребление сирот* - вести журнал на клиенте, после перезагрузки убить все осиротевшие процессы;

• *реинкарнация* – время разделено на эпохи и при перезагрузке клиент объявляет о начале новой эпохи;

• *мягкая реинкарнация* – при получении сообщения о смене эпохи каждая машина проверяет, есть ли на ней удаленные вычисления и пытается найти их владельца. Только если владелец не найден они уничтожаются;

• *истечение срока* – каждому действию дается какое-то время, если время истекло, а действие не закончено, то клиент должен явно продлить время.

Восстановление

Существует два основных способа восстановления после ошибок:

- *обратное исправление* (backward recovery)– возвращение из текущего ошибочного состояния в безошибочное. Для этого необходимо время от времени создавать контрольные точки (checkpoint).

Пример: повторная передача пакета.

- *прямое исправление* (forward recovery) – вместо отката делается попытка перевести систему в новое корректное состояние, в котором она смогла бы продолжать работать. Основная проблема состоит в том, что нужно заранее знать о возможных ошибках.

Пример: восстановление пропущенных данных на основе уже полученных.