МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное образовательное
учреждение высшего образования
ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

Институт математики, механики и компьютерных наук
имени И. И. Воровича

Направление подготовки
01.03.02 — Прикладная математика
и информатика

РЕАЛИЗАЦИЯ ОТКАЗОУСТОЙЧИВОГО ПРОСТРАНСТВА КОРТЕЖЕЙ СРЕДСТВАМИ ЯЗЫКА PYTHON

Выпускная квалификационная работа
на степень бакалавра

Студентки 4 курса
В. М. Ложкиной

Научный руководитель:
старший преподаватель кафедры информатики и вычислительного
эксперимента В. Н. Брагилевский

Допущено к защите:
руководитель направления ФИИТ _____ В. С. Пилиди

Ростов-на-Дону
2018

Содержание

Введение

В современном мире распределённые системы являются основой различных объектов, например, веб-сервисов и пиринговых сетей. Поскольку обмен сообщениями в системе осуществляется по ненадёжным каналам связи, сообщения, посылаемые системе или самой системой, могут просматриваться, перехватываться и подменяться. Это приводит к неправильному функционированию, отказу отдельных компонент или системы в целом. Поэтому в распределённых системах поддержке безопасности уделяется особое внимание.

Для повышения надёжности системы случайные и умышленные сбои интерпретируются как Византийские ошибки (в терминах задачи о византийских генералах). В этом случае можно использовать соответствующие отказоустойчивые техники, которые смогут защитить систему и от случайных сбоев, и от умышленных вторжений.

В данной работе рассматривается византийское пространство кортежей — открытая распределённая устойчивая к византийским ошибкам система, основанная на пространстве кортежей без распределённой памяти, в которой процессы взаимодействуют путём обмена сообщениями.

/здесь будет продолжение/

1. Распределённые вычислительные системы

Распределённая вычислительная система — это набор независимых компьютеров, реализующий параллельную обработку данных на многих вычислительных узлах. С точки зрения пользователя этот набор является единым механизмом, предоставляющим полный доступ к ресурсам. Существует возможность добавления новых ресурсов и перераспределения их по системе, возможность добавления свойст и методов, но информация об этих событиях скрыта от пользователя [**Tanenbaum**].

Распределённая система, рассматриваемая в данной работе, представлена множеством из n серверов. Взаимодействие клиентов с системой происходит с помощью вспомогательной промежуточной инфраструктуры, как показано на рисунке 1.

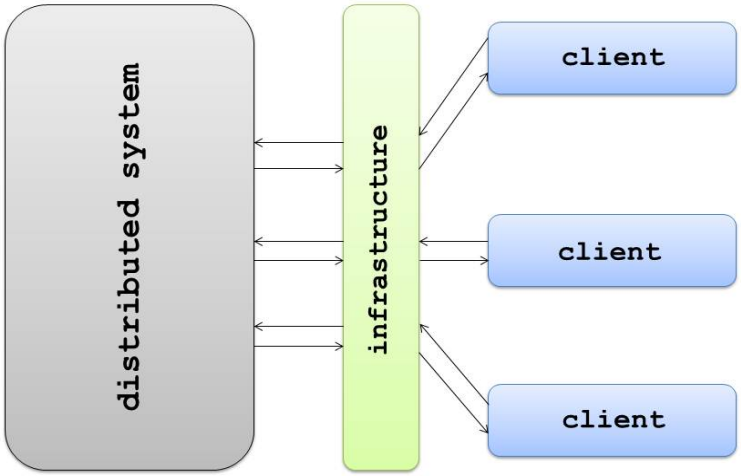

Рисунок 1 — Взаимодействие клиентов с распределённой системой

Одной из важнейших характеристик распределённых систем является отказоустойчивость. Отказоустойчивость — это свойство системы сохранять работоспособность в том случае, если какие-либо составляющие её компоненты перестали правильно функционировать.

Компоненты системы могут стать нероботоспособны по различным причинам, например, из-за технологических сбоев или атак безопасности.

Большинство современных распределённых систем имеют характеристики открытых систем. Открытая распределённая система предполагает использование служб, вызов которых требует стандартного синтаксиса и семантики. Такая система может иметь неизвестное количество ненадёжных и неоднородных участников, к тому же участникам не нужно быть активными одновременно (свойство разъединённости во времени) и не обязательно что-то знать друг о друге (свойство разъединённости в пространстве). Связь между узлами распределённой системы является ненадёжной (может прерываться, что повлечёт за собой потерю сообщений), обмен сообщениями может происходить не мгновенно, а с существенной задержкой. Кроме того, любой узел системы может отказать или быть выключен в любой момент времени. Все эти факторы неизбежно приводят к неправильному функционированию системы. Один из способов улучшить её надёжность — это интерпретировать случайные или умышленные неполадки как византийские ошибки (в терминах задачи византийских генералов), тогда использование отказоустойчивых техник сможет сделать координационную составляющую системы отказоустойчивой и для случайных сбоев, и для умышленных вторжений.

2. Задача византийских генералов

Задача византийских генералов — это задача взаимодействия нескольких удалённых абонентов, получивших сообщения из одного центра, причём часть этих абонентов, в том числе центр, могут быть предателями, то есть могут посылать заведомо ложные сообщения с целью дезинформирования. Нахождение решения задачи заключа-

ется в выработке единой стратегии действий, которая будет являться выигрышной для всех абонентов.

Формулировка задачи состоит в следующем. Византийская армия представляет собой объединение некоторого числа легионов, каждым из которых командует свой генерал, генералы подчиняются главнокомандующему армии Византии. Поскольку империя находится в упадке, любой из генералов и даже главнокомандующий могут быть заинтересованы в поражении армии, то есть являться предателями. Генералов, не заинтересованных в поражении армии, будем называть верными. В ночь перед сражением каждый из генералов получает от главнокомандующего приказ о действиях во время сражения: атаковать или отступать. Таким образом, имеем три возможных исхода сражения:

- Благоприятный исход: все генералы атакуют противника, что приведёт к его уничтожению и победе Византии.
- Промежуточный исход: все генералы отступят, тогда противник не будет побеждён, но Византия сохранит свою армию.
- Неблагоприятный исход: некоторые генералы атакуют противника, некоторые отступят, тогда Византийская армии потерпит поражение.

Так как главнокомандующий тоже может оказаться предателем, генералам не следует доверять его приказам. Однако если каждый генерал будет действовать самостоятельно, независимо от других генералов, то вероятность наступления боагоприятного исхода становится низкой. Таким образом, генералам следует обмениваться информацией между собой для того, чтобы прийти к единому решению.

На практике задача византийских генералов решается с помощью алгоритмов консенсуса, ярким представителем которых является алгоритм $Paxos$, предложенный Лесли Лампортом.

3. Paxos

$Paxos$ — это алгоритм для решения задачи консенсуса в сети ненадёжных вычислителей. Компоненты распределённой системы можно разделить на три группы:

- Заявитель ($Proposer$) — выдвигает «предложения» (какие-то значения), которые либо принимаются, либо отвергаются в результате работы алгоритма.
- Акцептор ($Acceptor$) — принимает или отвергает «предложение» Заявителя, согласует своё решение с остальными Акцепторами, уведомляет о своём решении Узнающих. Если Акцепторами было принято какое-либо значение, предложенное Заявителем, то оно называется утверждённым.
- Узнающий ($Learner$) — запоминает решение Акцепторов, принятое в результате работы алгоритма консенсуса.

Компоненты распределённой системы могут принадлежать сразу нескольким группам, описанным выше, и вести себя и как Заявитель, и как Акцептор, и как Узнающий. Такое распределение ролей в системе гарантирует следующее:

- Только предложенное Заявителем значение может быть утверждено Акцепторами.
- Акцепторами утверждается только одно значение из всех предложенных Заявителями значений (возможно, противоречивых).
- Узнающий не сможет узнать об утверждении какого-либо значения вплоть до того момента, пока оно действительно не будет утверждено.

  /здесь будет графическая интерпретация алгоритма/

4. Пространство кортежей

Кортеж — это структура данных, представляющая собой неизменяемый список фиксированной длины, элементы которого могут относиться к различным типам данных. Два кортежа считаются равными, если совпадают их длины, а также типы и значения соответствующих полей.

Хранилище кортежей, в котором доступ к элементам может осуществляться параллельно, называется пространством кортежей. Оно является основой языка программирования $Linda$. Данный язык предназначен для построения эффективных параллельных программ. Он включает в себя четыре операции манипулирования данными (кортежами):

- out — запись кортежа в пространство кортежей.
- rd — недеструктивное чтение кортежа.
- in — деструктивное чтение (извлечение) кортежа.
- $eval$ — создание нового процесса для обработки данных. Результаты работы этого процесса отразятся на общем пространстве.

Шаблоном кортежа будем называть кортеж, некоторые поля которого неопределены и не представляют важности. Кортеж соответствует шаблону, если длина кортежа равна длине шаблона и определённые в шаблоне поля совпадают по типу и значению с соответствующими полями кортежа.

Операция записи out принимает в качестве входного параметра кортеж, все поля которого определены. Операции чтения rd и in принимают в качестве входного параметра шаблон кортежа, по которому производится поиск соответствующих кортежей в пространстве. Кроме того, операции чтения являются блокирующими, то есть если в пространстве кортежей на данный момент нет ни одного кортежа, соответствующего шаблону, то процесс, пославший запрос, заблокиру-

ется до того момента, пока в пространстве не появится подходящего кортежа.

Таким образом, пространство кортежей можно рассматривать как разновидность распределённой памяти: например, одна группа процессов записывает данные в пространство, а другая группа извлекает их и использует в своей дальнейшей работе.

Для того, чтобы поиск кортежей в пространстве кортежей занимал минимальное время, адресация в нём осуществляется по содержимому (например, с помощью алгоритмов хэширования). Иными словами, пространство кортежей — реализация парадигмы ассоциативной памяти.

В данной работе рассматривается распределённая система под названием «Византийское пространство кортежей». Основой системы является пространство кортежей, имеющее ряд особенностей, которые позволяют добиться отказоустойчивости.

5. BTS: Византийское пространство кортежей

Системная модель византийского пространства кортежей предполагает бесконечное число процессов-клиентов $\Pi = \{p_1, p_2, \dots\}$, которые взаимодействуют со множеством из n серверов $U = \{s_1, s_2, \dots, s_n\}$, что симулирует пространство кортежей. Клиенты и серверы коммуницирует с помощью обмена сообщениями.

Будем полагать, что случайное число клиентов и связка серверов из $f \leqslant \left\lfloor \frac{n-1}{3} \right\rfloor$ штук могут быть подвержены византийским ошибкам: они могут произвольным образом отклоняться от их спецификаций и работать в сговоре, чтобы изменить поведение системы. Такие процессы будем называть неисправными, а правильно работающие процессы — корректными.

Византийское пространство кортежей — это ассиметричная кворум-система, представляющая собой множество кворумов серверов $\mathcal{L} \in 2^U$, в котором каждая пара кворумов из $\mathcal{L}$ пересекается на достаточно многих серверах и всегда есть кворум со всеми корректными серверами. Существование пересечений между кворумами позволяет совершенствовать протоколы чтения/записи, позволяя поддерживать целостность разделённой переменной даже если эти операции были выполнены в разных кворумах системы.

Из-за ассиметричности кворум-системы будем полагать, что $n > 3f + 1$. Сами кворумы делятся на два типа ($\mathcal{L} = \mathcal{L}_{\triangledown} \cup \mathcal{L}_{\sqsupseteq}$):

- чтение ($Q_r \in L_r$), состоят из $\left\lceil \frac{n+f+1}{2} \right\rceil$ серверов,
- запись ($Q_w \in L_w$), состоят из $\left\lceil \frac{n+f+1}{2} \right\rceil + f$ серверов.

Теперь опишем протоколы чтения/записи из языка $Linda$, но сделаем операции чтения неблокирующими. Положим, что каждый сервер $s \in U$ имеет локальную копию пространства кортежей T_s и множество кортежей для удаления R_s, при этом идентичных кортежей не существует. Тогда стандартные операции над множествами могут применяться к этим двум множествам кортежей.

5.1. Операция записи out

Вызов функции $out(t)$ добавляет кортеж в пространство. На стороне клиента эта операция реализуется с помощью алгоритма 1:

Алгоритм 1. Операция out

```
   {client p}
 1: procedure out(t)
 2:     for all s ∈ Q_w do
 3:         send(s, ⟨ OUT, t⟩)
 4:     end for
 5: end procedure
```

При вызове данной функции клиенту p не нужно ждать ответа от серверов. Будем считать, что операция завершится в тот момент, когда все корректные серверы из кворума записи получат кортеж t.

На стороне сервера операция out реализуется с помощью алгоритма 2:

Алгоритм 2. Операция out

```
   {server s}
1: upon receive(p, ⟨ OUT, t⟩)
2:     if t ∉ R_s then
3:         T_s ← T_s ∪ {t}
4:     end if
5:     R_s ← R_s \ {t}
6: end upon
```

При получении запроса $\langle$ OUT, $t\rangle$ от клиента p сервер добавляет кортеж t в пространство кортежей только в том случае, если этот кортеж ранее не был из него удалён. Это необходимо для того, чтобы один и тот же кортеж не был удалён дважды.

5.2. Операция чтения rdp

Недеструктивная операция чтения rdp в качестве входного параментра принимает шаблон кортежа $\bar{t}$, возвращает копию соответствующего шаблону кортежа из пространства. На стороне клиента эта операция реализуется с помощью алгоритма 3:

Алгоритм 3. Операция rdp

```
    {client p}
 1: function rdp(t̄)
 2:     T[s_1, ..., s_n] ← (⊥, ..., ⊥)
 3:     for all s ∈ U do
 4:         send(s, ⟨ RD, t̄⟩)
 5:     end for
 6:     repeat
 7:         wait receive(s, ⟨ TS, T_s^t̄⟩)
 8:         T[s] ← T_s^t̄
 9:     until {s ∈ U : T[s] ≠⊥} ∈ L_▽
10:     if ∃t : (|s : t ∈ T[s]| ⩾ f + 1) then
11:         return t
12:     end if
13:     return ⊥
14: end function
```

Клиент p взаимодействует с кворумом серверов чтения, в качестве ответа на запрос он получает от каждого сервера список подходящих под шаблон кортежей (серверная часть реализуется с помощью алгоритма 4), затем клиент выбирает из них общий кортеж t, который располагается как минимум на $f+1$ сервере. Если такого кортежа нет, то возвращается спецсимвол $\perp$, обозначающий, что операция не увенчалась успехом. Заметим, что операция rdp не является блокирующей в отличие от её аналога rd в языке $Linda$.

Алгоритм 4. Операция rdp

```
    {server s}
1: upon receive(p, ⟨ RD, t⟩)
2:     T_s^t̄ ← {t ∈ T_s : m(t, t̄)}
3:     send(p, ⟨ TS, T_s^t̄⟩)
4: end upon
```

5.3. Операция чтения inp

Операция inp является операцией деструктивного чтения, она имеет более сложную реализацию, поскольку один кортеж не может быть удалён из пространства двумя различными вызовами inp. Этот факт подразумевает использование критических секций для процессов, пытающихся удалить один и тот же кортеж, что, во-первых, обеспечит невозможность удаления кортежа двумя процессами одновременно, во-вторых, позволит всем процессам получить доступ к ресурсу в некоторое время. Операции входа в критическую секцию и выхода из неё реализуются через алгоритмы 3 и 4 соответственно:

Алгоритм 5. Операция enter

```
    {client p}
 1: function enter(t̄)
 2:     T[s_1, ..., s_|Q_r|] ← (⊥, ..., ⊥)
 3:     TO-multicast(U, ⟨ ENTER, p, t̄⟩)
 4:     for all s ∈ Q_r do
 5:         wait receive(s, ⟨ GO, p, T_s^t̄⟩)
 6:         T[s] ← T_s^t̄
 7:     end for
 8:     if ∃t : (|s : t ∈ T[s]| ⩾ f + 1) then
 9:         return t
10:     end if
11:     return ⊥
12: end function
```

Алгоритм 6. Операция exit

```
    {client p}
1: procedure exit(t̄)
2:     for all s ∈ U do
3:         wait send(s, ⟨ EXIT, p, T_s^t̄⟩)
4:         T[s] ← T_s^t̄
5:     end for
6: end procedure
```

Алгоритм 7. Операция inp

```
    {client p}
 1: function inp(t̄)
 2:     repeat
 3:         t ← enter(t̄)
 4:         if t =⊥ then
 5:             exit(t̄)
 6:             return ⊥
 7:         end if
 8:         d ← paxos(p, P, A, L, t)
 9:         exit(t̄)
10:     until d = t
11:     return t
12: end function
```

Для того, чтобы удалить кортеж, соответствующий шаблону $\bar{t}$, из пространства, серверы из кворума чтения должны принять решение, какой именно кортеж удалять. Для этой цели используется $Paxos$ — алгоритм достижения консенсуса в распределённых системах. Операция деструктивного чтения inp реализуются через алгоритмы 5: /здесь будет псевдокод алгоритма 5/

В реализации операции inp алгоритм $Paxos$ приводится в действие путём вызова функции $paxos$, которая имеет пять параметров:

1. Процесс p, предлагающий значение. В нашем случае это клиентский процесс, вошедший в критическую секцию.

2. Множество Заявителей $P = \{p, s_1, ..., s_{f+1}\}$, состоящее из клиентского процесса p и $f + 1$ сервера. Такой набор гарантирует наличие хотя бы одного корректного заявителя.

3. Множество Акцепторов $A = U$. Все серверы являются Акцепторами.

4. Множество Узнающих $L = \{p\} \cup U$. О принятом решении узнают все серверы и клиентский процесс.

5. Предлагаемое значение t.

На момент выхода из функции $paxos$, все Акцепторы уже приняли решение, удалять кортеж t или нет, все Узнающие получили сообщения о решении Акцепторов. Если функция $paxos$ вернула кортеж, то он уже был удалён из пространства кортежей, иначе возвращается спецсимвол $\perp$, сигнализирующий о том, что операция удаления кортежа t не увенчалась успехом. Данный результат иллюстрирует благоприятный и промежуточный исходы задачи византийских генералов. При благоприятном исходе из пространства кортежей удаляется кортеж, предложенный клиентом, то есть приказ главнокомандующего (клиента) был получен и изучен всеми генералами (серверами), после чего они единогласно приняли решение следовать приказу. При промежуточном исходе кортеж не удаляется, возвращается спецсимвол $\perp$, при этом сохраняется целостность пространства кортежей, поскольку на всех корректных серверах удаления не произошло. Иными словами, генералами (серверами) было принято единогласное решение не доверять приказу главнокомандующего (клиента) и отступить (не производить удаление), сохранив при этом армию (целостность пространства кортежей). /здесь будет продолжение/

5.4. Блокирующие операция rd и in

Блокирующие операции rd и in, соответствующие спецификации языка $Linda$, могут быть реализованы на стороне клиента путём повторного вызова их неблокирующих аналогов rdp и inp до тех пор, пока необходимый кортеж не будет получен.