МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное образовательное
учреждение высшего образования
ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

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

Направление подготовки
Прикладная математика и информатика

Кафедра информатики и вычислительного эксперимента

РЕАЛИЗАЦИЯ ИНТЕРВАЛЬНОГО ВРЕМЕНИ В RABBITMQ

Выпускная квалификационная работа
на степень бакалавра студентки
А. А. Мухаррам

Научный руководитель:
ст. преп. В. Н. Брагилевский

Ростов-на-Дону
2015

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Отсутствие глобального соглашения о времени может привести к возникновению ошибок в системе. В таких случаях принято говорить о необходимости синхронизации часов. Синхронизация часов связана с пониманием порядка следования во времени событий конкурентных процессов. Она может быть полезна в тех случаях, когда необходимо синхронизировать обмен сообщениями, контролировать совместное использование ресурсов и выполнение совместной работы несколькими процессами. Таким образом, задача синхронизации в распределенной системе - обеспечить возможность принятия согласованного решения процессами о порядке следования событий.

Одним из вариантов решения данной проблемы является использование логических часов. Этот механизм впервые был предложен и реализован Лэсли Лампортом в 1978 году. В своей статье [1] он указал, что обычно имеет значение не точное время выполнения события процесса, а его порядок.

Лампорт определил логические часы как способ присвоить номер событию, где под номером понимается время, в которое наступило событие. Для каждого процесса P_i определяются часы C_i как функция, которая ставит в соответствие каждому событию a процесса P_i число $C_i(a)$. Система в целом описывается с помощью функции C, которая ставит в соответствие каждому событию b число $C(b)$, где $C(b) = C_j(b)$, если b - событие процесса P_j. С помощью этой функции можно сравнивать время наступления событий в распределенной системе.

Интервальные отметки времени - один из алгоритмов логических часов. В данной работе описан процесс разработки плагина для брокера сообщений RabbitMQ, реализующего мониторинг событий брокера на основе интервальных отметок времени. С помощью данного плагина можно восстановить цепочку событий между любыми двумя событиями системы. Под событием в RabbitMQ будет подразумеваться отправка или получение сообщения.

ГЛАВА 1

ЛОГИЧЕСКИЕ ЧАСЫ

Эта глава будет посвящена рассмотрению синхронизации процессов посредством нескольких алгоритмов логических часов.

Введем основные понятия. Для синхронизации логических часов Лампорт определил не рефлексивное, транзитивное отношение под названием «происходит раньше», которое удовлетворяет следующим 3 условиям:

- Если a и b — события, происходящие в одном и том же процессе, и a происходит раньше, чем b, то отношение $a \to b$ истинно.
- Если a - отправка сообщения одним процессом, а b - это получения этого сообщения другим процессом, то отношение $a \to b$ истинно.
- Если $a \to c$ и $c \to b$, тогда из $a \to b$

Два отдельных события a и b конкурентные, если оба отношения $a \to b$ и $b \to a$ несправедливы.

Основываясь на введенном отношении и функции логических часов, запишем условие: для любых двух событий a и b, если $a \to b$ истинно, то $C(a) < C(b)$. Ясно, что это условие выполняется, если события a и b удовлетворяют одному из условий перечисленных выше.

Рассмотрим последовательность событий, происходящих между тремя процессами, изображенную на рисунке 1.1. Процессы запущены на разных машинах, каждая из которых имеет собственные

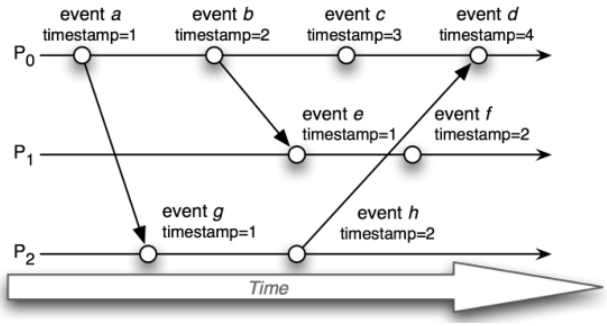

Рисунок 1.1 — неупорядоченная последовательность событий

часы и скорость работы. Каждый процесс поддерживает свой глобальный счетчик, который увеличивается на единицу перед тем, как присвоить новому событию временную отметку. Анализируя временные отметки событий, изображенных на рисунке 1.1, можно заметить несколько особенностей. Событие g, событие изображающее получение сообщения, посланного событием a, имеет такую же временную отметку, что и событие a, хотя совершенно ясно, что оно произошло после события a. Событие e имеет временную отметку меньше, чем событие отправившее ему сообщение (событие b).

1.1. Временные отметки Лампорта

Алгоритм Лампорта исправляет ситуацию, описанную выше, перенумеровывая временные отметки так, чтобы для событий, относящихся к отправке и получению сообщений, выполнялось отношение «происходит раньше».

Правила:

- Каждый процесс имеет счетчик, который увеличивается на единицу перед каждым внутренним событием процесса. События-ми процесса считается получение и отправка сообщений.
- К сообщению при отправке прикрепляется значение счетчика процесса.

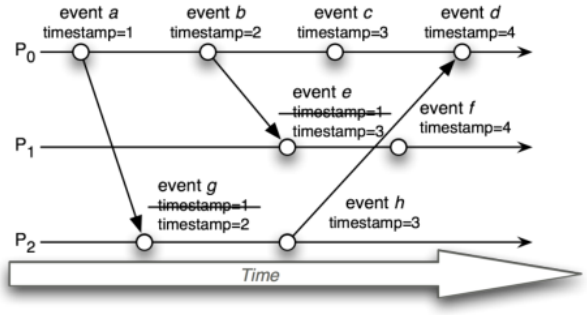

Рисунок 1.2 — упорядоченная последовательность событий

- При получении сообщения, если значение счетчика процесса-получателя меньше временной отметки полученного сообщения, процесс-получатель меняет значение счетчика на значение полученной временной отметки, иначе ничего не меняется.

Применив этот алгоритм к последовательности сообщений, изображенной на рисунке 1.1, мы получим правильно упорядоченный поток сообщений среди событий, связанных причинно-следственной связью (рисунок 1.2). Каждое событие системы теперь имеет временную отметку, которая называется временной отметкой Лампорта. В условиях выполнения правил данного алгоритма, для любых двух событий системы a и b , если отношение $a \to b$ истинно, то $L(a) < L(b)$, где $L(x)$ - временная отметка Лампорта для события x.

1.2. Векторные отметки времени

Благодаря алгоритму Лампорта события в системе имеют единую последовательность, однако из сравнения временных отметок Лампорта $L(a)$ и $L(b)$ нельзя сделать вывод о взаимосвязи между событиями a и b. Причинно-следственная связь может быть установлена посредством векторных отметок времени (vector timestams).Этот алгоритм был независимо предложен Маттерном в 1989 г.(Mattern) и Фиджем в 1991 г. (Fidge).

Векторная отметка времени в системе из N процессов - целочисленный вектор длины N.

Правила для использования векторных отметок времени:

- Каждый процесс в системе имеет свою локальную копию вектора (вектор V_i для процесса P_i), которая инициализируется 0: $V_i[j] = 0, i, j = 1..N$
- Процесс P_j перед каждым новым внутреннем событием увеличивает свою компоненту вектора на единицу: $V_j[j] = V_j[j] + 1$
- Сообщение отправляется процессом P_i вместе с векторной отметкой времени V_i
- При получении сообщения процесс P_j сравнивает полученную временную отметку t со своим локальным вектором поэлементно, устанавливая каждую компоненту локальной временной отметки как максимум из двух значений: $V_j[i] = max(V_j[i], t[i]), i = \overline{1, N}$

Сравниваются векторные отметки по определению:

$$V = V', \text{ если } V[i] = V'[i],\ i = \overline{1, N}$$
$$V \leq V', \text{ если } V[i] \leq V'[i],\ i = \overline{1, N}$$

С помощью алгоритма векторных часов получаем следующие утверждения:

Если отношение $a \to b$ истинно, то $V(a) < V(b)$

Если $V(a) < V(b)$, то отношение $a \to b$ истинно

Два события a и b называются конкурентными, если высказываение $V(a) \leq V(b)$ or $V(b) \leq V(a)$ ложно

Рассмотрим последовательность событий, изображенную на рисунке 1.3. Можно заметить, что события a и e конкурентные,т.к. не каждый элемент одного вектора меньше или равен соответствующему элементу другого, а события b и c взаимосвязаны.

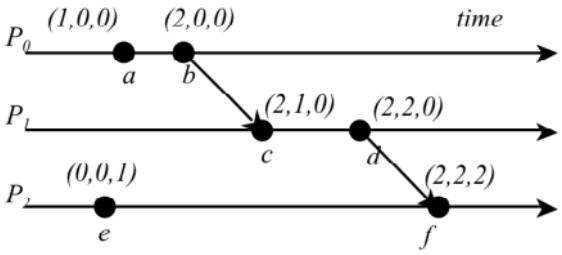

Рисунок 1.3 — векторный алгоритм

1.3. Счетчики дерева интервалов: Логические часы для динамических систем

Хотя отслеживание взаимосвязей между событиями в системах с динамическим изменением числа участников возможно с помощью модифицированного алгоритма векторных часов, в большинстве таких алгоритмов наблюдается чрезмерный структурный рост, а также локализованное удаление не является поддерживаем процессов. Решение этих проблем возможно с использованием алгоритма счетчиков дерева интервалов (Interval Tree clocks).

Данному механизму не требуются глобальные идентификаторы, он способен автономно создавать, удалять и повторно использовать их без помощи глобальной координации; любой объект может создать дочерний объект и количество участников может быть сокращено путем присоединения к произвольным парам объектов; метки могут расти или сокращаться, адаптируясь к динамической природе системы.

Механизм отслеживания причинно-следственной связи для алгоритма интервального времени моделируется посредством ряда основных операций: порождения нового процесса($fork$), события($event$) и слияния($join$), воздействующих на интервальные отметки времени (логические часы), чья структура представляет собой пару (i,e), образованную идентификатором и событием, в котором и содержатся все возможные причинно-следственные связи.

Основная идея алгоритма заключается в том, что идентификатор каждого участника является множеством интервалов, которые используются для увеличения компоненты событие при наступлении внутреннего события процесса и для передачи последующим элементам при порождении нового процесса.

Рассмотрим операции над интервальными отметками времени. При выполнении операция **fork** сохраняется компонента событие, а идентификатор делится на два непересекающихся интервала: $fork(i,e) = ((i_1,e),(i_2,e))$ Операция $peek$ - частный случай операции $fork$: $peek(i,e) = ((0,e),(i,e))$

Операция **event** добавляет новое событие к временной отметке (не анонимной, $(0,e)$) так, что, если $event(i,e) = (i,e')$, то $e < e'$. Обнаружение причинно-следственной связи на основе интервальных отметок осуществляется посредством сравнения компонент событие.

Операция **join** осуществляет слияние двух отметок: $join((i_1,e_1),(i_2,e_2)) = (i_3,e_3)$, где $e_3 > e_2$, $e_3 > e_1$, $e_3 = e_2 \sqcup e_1$, $i_3 = f(i_2,i_3)$, i_3 уникален на уровне системы.

Классические операции отправки, получения и синхронизации реализуются как композиция основных операций:

$$send = peek \circ event$$
$$receive = event \circ join$$
$$sync = fork \circ join$$

В ITC используется исходная метка ($seed$), $(1,0)$, из которой можно получить необходимое число участников N с помощью применения к ней операции $fork$ N раз. Рассмотрим пример, изображенный на рисунке 1.4. В ITC используется графическая нотация, нижний слой отметки времени - изображение идентификатора, верхние - событие. Работа алгоритма начинается с единственного участника ($seed$) с исходной меткой, которая преобразовывается в две временные отметки под действием операции fork. На данный момент в системе два участника. В поддереве участника, соответсвующего верхней ветке дерева

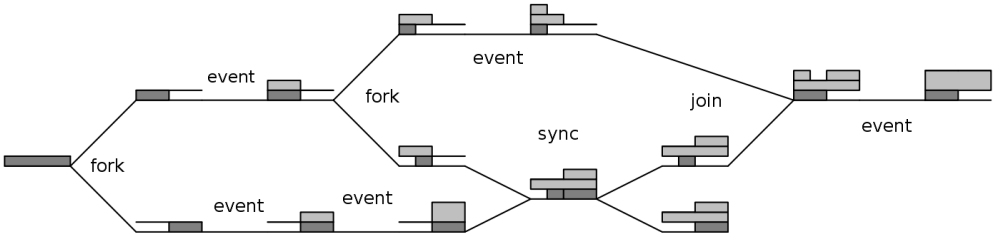

Рисунок 1.4 — интервальные отметки времени

интервалов,происходит внутреннее событие с последующим за ним порождением нового процесса ($fork$). В поддереве участника, соответствующего нижней ветке, происходит два новых события. В этот момент число участников выросло до трех. Затем один из участников подвергается действию события, в то время как оставшиеся два участника синхронизируются. В результате с помощью операции $join$ происходит объединением двух поддеревьев. Этот пример показывает, как просто ITC адаптируется к числу участников системы.