Московский государственный университет имени М.В. Ломоносова

Алгоритмы синхронизации времени в Unix-системах

Автор: Шпилевой Владислав Дмитриевич

Научный руководитель: Александр Герасев

Москва, 2015

Аннотация

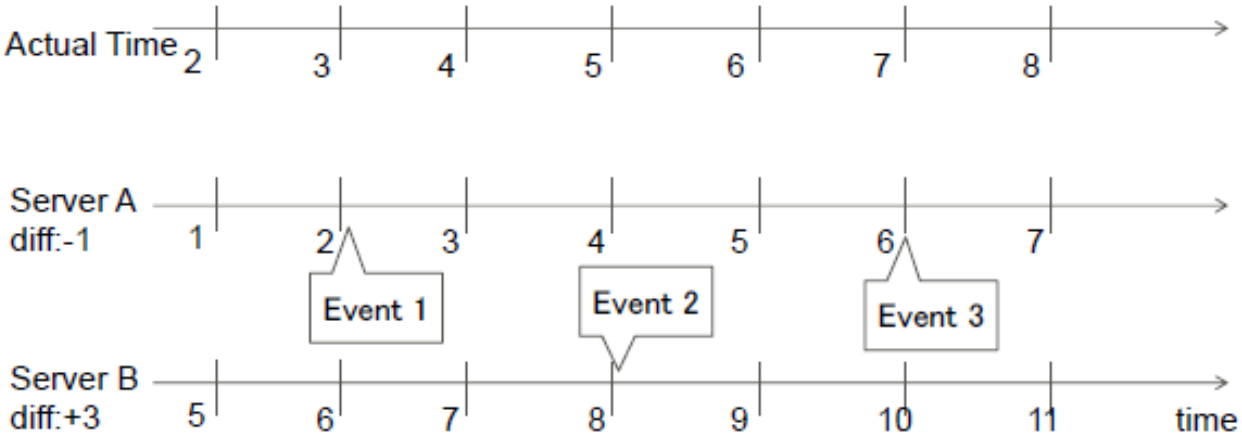
В данной работе описаны некоторые из существующих подходов к решению задачи синхронизации времени в распределенных Unix-системах и краткие обзоры их реализаций, а так же описание нового алгоритма на их основе. Основной задачей стоит синхронизация счетчиков времени между основной системой и подключенными к ней устройствами.

Оглавление

Введение

С чего вообще появилась необходимость решать подобную задачу?

Любой компьютер имеет механизм подсчета времени. Это то, что мы называем часами, хотя это скорее таймер, который реализован следующим образом: в процессоре есть кристалл, который под напряжением колеблется с определенной частотой, которая зависит от загруженности системы, новизны и надежности оборудования, температуры воздуха, загруженности сети, свойств конкретного кристалла, подаваемого напряжения и т.д. Колебания вызывают изменения счетчика, ответственного за формирование системного времени. Таким образом, из-за различной частоты колебания кристалла имеет место рассинхронизация времени в распределенной системе. В связи с этим в ней могут возникать ситуации вроде этой:



Event Ordering based on Actual Time

Time	Event
3	Event 1
5	Event 2
7	Event 3

Event Ordering based on Timestamp

Time	Event	
2	Event 1	
6	Event 3	← reverse!
8	Event 2	←

То есть метки времени, соответствующие событиям, противоречат их действительному порядку.

В данной работе рассматриваются алгоритмы, позволяющие таких ситуаций избежать или хотя бы существенно уменьшить их количество, синхронизируя работу отдельных узлов системы.

Целью данной работы является построение нового алгоритма синхронизации, который предполагается применять для синхронизации времени между подчиненными устройствами и основной системой. Так же в рамках этой работы данный алгоритм был реализован, а как именно, описано далее.

Обзор существующих алгоритмов

Так или иначе все алгоритмы должны обеспечивать выполнение основных требований:

- в определенные моменты системное время узлов должно совпадать максимально
- часы не должны идти назад (но они могут быть замедлены или ускорены)

Алгоритм Кристиана

Краткое описание

Есть эталонный сервер, откуда все зависимые узлы запрашивают время. Сам эталонный сервер, тем временем, должен отдавать время UTC, для чего он сам должен быть подключен к некоему серверу, которому это время известно. Так же в этом алгоритме учтен важный момент - сообщения с временными метками не доходят мгновенно. Для оценки времени передачи по сети предлагается использовать половину времени между отправкой запроса и получением ответа.

Подробнее

Периодически, гарантировано не реже, чем каждые $\delta/2\rho$ секунд, где δ – установленный для узла максимальный дрейф времени, ρ – технически определённый дрейф часов, каждый узел посылает серверу времени сообщение, запрашивая текущее время. Сервер отвечает сообщением, содержащим его текущее время.

В качестве первого приближения, когда отправитель получает ответ, он может просто выставить свои часы в полученное значение. Однако такой алгоритм имеет две проблемы: **главную** и **второстепенную**.

1) Главная проблема состоит в том, что время никогда не течет назад. Если часы отправителя спешат, то полученное от сервера время может оказаться меньше текущего значения времени у отправителя. Простая подстановка времени сервера способна вызвать серьезные проблемы, связанные с тем, например, что объектные файлы, скомпилированные после того, как было изменено время, становятся помечены временем более ранним, чем модифицированные исходные тексты, которые поправлялись до изменения времени, что может привести к невозможности дальнейшей трансляции исходных кодов.
Для решения этой проблемы изменения могут вноситься постепенно. Предположим, что таймер узла, на котором производится синхронизация времени, настроен так, что он генерирует 100 прерываний в секунду. В нормальном состоянии каждое прерывание будет добавлять ко времени по 10 мс. При запаздывании часов сервера процедура прерывания будет добавлять каждый раз всего по 9 мс. Соответственно, часы узла должны быть замедлены так, чтобы добавлять при каждом прерывании 9 мс.

2) Менее серьезная проблема состоит в том, что перенос ответного сообщения с сервера времени отправителю требует ненулевого времени. Метод решения проблемы состоит в измерении этой величины. Отправителю достаточно просто аккуратно записать интервал между посылкой запроса и приходом ответа. То и другое время измеряется по одним и тем же часам, а значит, интервал будет относительно точно измерен даже в том случае, если часы отправителя имеют некоторое расхождение с UTC. В отсутствии какой-либо дополнительной информации наилучшим приближением времени прохождения сообщения будет половина разницы между временем отправки запроса и получения

ответа. После получения ответа, чтобы получить приблизительное текущее время сервера, значение, содержащееся в сообщении, следует увеличить на это число. Эта оценка может быть улучшена, если приблизительно известно, сколько времени сервер времени обрабатывает прерывание и работает с пришедшим сообщением.

Для повышения точности Кристиан предложил производить не одно измерение, а серию. Все измерения, в которых время доведения превосходит некоторое пороговое значение, отбрасываются как ставшие жертвами перегруженной сети, а потому недостоверные. Оценка делается по оставшимся замерам, которые могут быть усреднены для получения наилучшего значения. С другой стороны, сообщение, пришедшее быстрее всех, можно рассматривать как самое точное, поскольку оно предположительно попало в момент наименьшего трафика и потому наиболее точно отражает чистое время прохождения.

Достоинства:

- время для передачи сообщения по сети частично учитывается и компенсируется
- очень легкая реализация
- достаточно эффективен в небольших системах

Недостатки:

- передача по сети занимает неопределенное время, и застраховаться от него лишь средним арифметическим от отправки и получения нельзя - пока сообщение будет подготовлено для передачи по сети, пока пакеты дождутся очереди для отправки, пока дойдут, пока будут распакованы и получены прикладной программой, может пройти немало времени. Плюс это время всегда будет колебаться, завися от нагрузки на сеть, ее надежности, современности оборудования. Например, если пакет будет проходить через маршрутизатор, то там он может задержаться на достаточно долгое время, буквально на миллисекунды, а может сразу отправиться дальше
- требуется наличие некоего главного сервера, а значит при его выходе из строя все система будет разрушена
- нет страховки от нарушения второго обязательного требования к синхронизации

Алгоритм Беркли

Краткое описание

Нередки ситуации, когда узнать эталонное время нельзя. Как раз для таких случаев предназначен алгоритм Беркли.

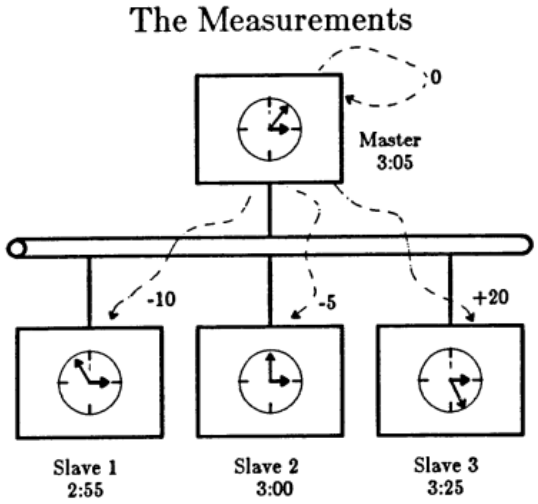
Есть некий сервер времени, который периодически опрашивает все узлы об их времени, усредняет полученные значения, и рассылает полученное время обратно для установки локальных часов узлов в это значение или их замедления.

Подробнее

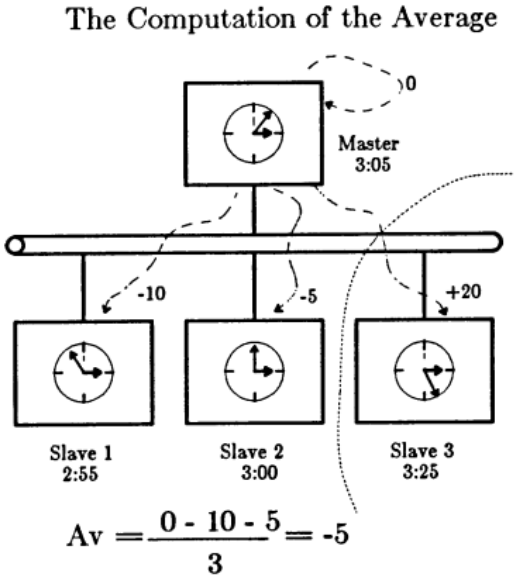
На каждом узле системы работает демон времени. Среди всех узлов есть один главный, который и занимается коррекцией системного времени.

Что происходит, когда в сеть подключается новый узел? - он запрашивает у главного сервера время, устанавливает его себе и становится равноправной частью системы.

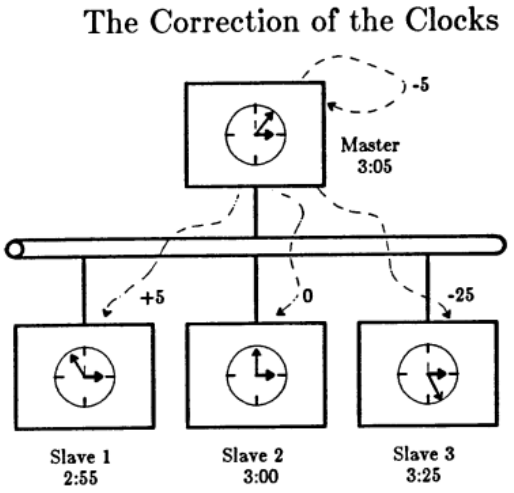
Через некоторые заранее заданные промежутки времени главный сервер опрашивает все узлы сети об их времени и запоминает, на сколько каждый узел отклоняется от главного времени.

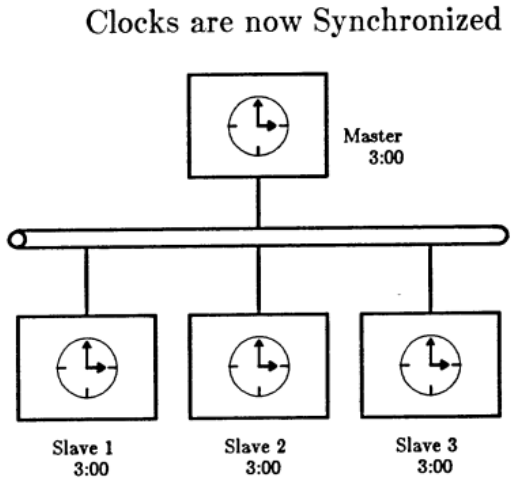


Затем происходит вычисление среднего отклонения (самые большие отклонения не учитываются, как ошибочные)



После чего оно добавляется к часам главного сервера, и всем узлам производится рассылка нового времени:





Достоинства:

- легкость в реализации
- высокая эффективность для систем, некритичных к отклонениям времени
- не требуется "связь с внешним миром" для получения каких-то внешних меток времени

Недостатки:

- требуется наличие некоего главного сервера, а значит при его выходе из строя все система будет разрушена
- так как время считается именно локально, независимо от других источников, то время в такой системе может оказаться далеко от реального
- "плохие" узлы (медленные, неустойчивые по частоте) оказываются влияние на общее время. То есть нет защиты от того, что резкий скачок на одном из узлов испортит среднее

Усредняющие алгоритмы

Краткое описание

Семейство алгоритмов с одним основополагающим принципом работы. С заданной периодичностью все узлы системы производят рассылку текущего времени. Затем в течение определенного времени принимают сообщения о текущем времени от других узлов. Когда все сообщения приняты, запускается алгоритм вычисления текущего времени. Простейший вариант – усреднение полученных значений. Известно несколько способов усовершенствования алгоритма.

Подробнее

Оба ранее описанных алгоритма сильно централизованы с неизбежно вытекающими отсюда недостатками. Известны также и децентрализованные алгоритмы. Один из классов алгоритмов децентрализованной синхронизации часов работает на основе деления времени на синхронизационные интервалы фиксированной продолжительности. В начале каждого интервала каждый узел производит широковещательную рассылку значения текущего времени на своих часах. Поскольку часы на различных машинах идут с немного различной скоростью, эти широковещательные рассылки будут сделаны не одновременно. После рассылки машиной своего времени она запускает локальный таймер и начинает собирать все остальные широковещательные пакеты в течение некоторого интервала. Когда будут собраны все широковещательные пакеты, запускается алгоритм вычисления по ним нового времени. Простейший алгоритм состоит в усреднении значений всех остальных машин.

Существуют способы усовершенствования алгоритма:

1) отбрасывать самые большие и самые маленькие значения и усреднять оставшиеся. Отбрасывание крайних значений можно рассматривать как самозащиту от неправильных часов.

2) пытаться скорректировать каждое из сообщений, добавляя к ним оценку времени прохождения от источника. Эта оценка может быть сделана на основе знания топологии сети или измерением времени прохождения эха.

Достоинства:

- выход из строя одного или нескольких узлов не разрушит систему, а лишь изменит общий вклад в среднее значение
- при усовершенствовании указанными способами может оказаться довольно надежным

Недостатки:

- сильная зависимость эффективности от загрузки сети
- низкая степень синхронизации
- не обеспечивают установки на всех узлах одинакового времени (часть сообщений на конкретном узле может не быть получена, не получена вовремя, отброшена как заведомо ложная)

Отметки времени Лампорта

Краткое описание

При обмене данными между взаимодействующими узлами происходит также обмен сведениями об их локальном времени. Если отметка времени, указанная в сообщении, оказывается меньше, чем локальное время узла, то все в порядке. Иначе получается, что сообщение было послано позже, чем получено, и делается вывод о необходимости коррекции системного времени узла, и время выставляется в значение, равное отметке в сообщении плюс единица.

Подробнее

Отметки Лампорта - простой алгоритм для определения порядка возникновения событий в распределенной системе. Алгоритм требует минимальных ресурсов, и так же служит основой для другого алгоритма - *"Vector Clocks"*.

Распределенные алгоритмы часто служат для синхронизации доступа к каким-либо ресурсам. Например, рассмотрим систему из двух процессов и диска. Процессы отправляют друг другу сообщения, а так же запрашивают доступ к диску. Диск дает доступ в том порядке, в котором приходят запросы. Допустим процесс 1 отправил диску запрос на запись и сообщение второму процессу, чтобы тот прочитал информацию с диска. Процесс 2 получает сообщение и так же обращается к диску для чтения, но из-за различных задержек в системе запросы пришли к диску одновременно - как ему определить, кому дать доступ? Здесь важен именно порядок запросов, а не конкретно значения часов, как в других алгоритмах.

Отметки времени Лампорта позволяют определить порядок с помощью счетчиков, которые есть в каждом процессе. Счетчик работает следующим образом:

1) Процесс увеличивает свой счетчик перед каждым событием, которое в нем возникает.

2) Когда процесс отправляет сообщение, он прикрепляет к нему текущее значение счетчика.

3) Когда процесс получает сообщение, он устанавливает свой счетчик в величину **max(**<текущее значение счетчика>, <значение счетчика в пришедшем сообщении>**) + 1**.

Вообще, этот счетчик можно рассматривать как некие логические часы, которые предназначены только для сообщений, перемещаемых между процессами. Фактически, когда процесс получает сообщение, он синхронизирует себя с отправителем.

Но у этого алгоритма существуют особенности в работе. Отметки Лампорта могут обеспечивать только **частичный** порядок событий между процессами. Введем обозначение *"событие A случилось до события B"*: A → B. Тогда условие непротиворечивости счетчика **C** можно сформулировать так:

Если A → B, то C(A) < C(B)

Сильное условие непротиворечивости формулируется так:

A → B *тогда и только тогда, когда* C(A) < C(B)

Вернувшись теперь к отметкам Лампорта, стоит запомнить, что этот алгоритм обеспечивает только простое *условие непротиворечивости*, но не *сильное*.

Достоинства:

• выход из строя одного или нескольких узлов не разрушит систему, и даже более того, не повлияет на время остальных узлов

• высокая эффективность при малом числе узлов

• высокая степень синхронизации (время переводится только вперед, и нет необходимости ждать эффекта от замедления на спешащих узлах)

Недостатки:

• не принимается в расчет время передачи сообщений между узлами

• общесистемное время скорее всего имеет мало общего с реальны

• сильное условие непротиворечивости часов не гарантируется к выполнению

Обзор построенного алгоритма

Общий принцип работы

В системе есть модуль, называемый далее подсистемой, который выступает в роли некоего управляющего сервера для все подчиненных узлов. Узлами называются другие модули, драйвера.

Каждый подчиненный должен подключать к себе заголовочный файл, где определены функции для взаимодействия с подсистемой. Новый узел должен зарегистрироваться в подсистеме, заполнить структуру с информацией о себе, и он сможет транслировать свое локальное время в системное вызовом одной функции.

То есть алгоритм напоминает алгоритм Беркли, но только подсистема не делает регулярной рассылки времени по всей системе.

Что должен сделать подчиненный узел

Новый узел вызывает функцию из заголовочного файла, которой он регистрирует себя в подсистеме. При регистрации узел должен указать о себе некоторую информацию и передать ее в структуре, чтобы подсистема эффективнее пересчитывала его время. Необходимым условием регистрации является определение функции, которая должна возвращать локальное время узла (значение счетчика времени). После регистрации узел может в любой момент вызывать функцию, конвертирующую его локальное время в системное. Когда узлу больше нет необходимости использовать подсистему, он должен удалить себя из подсистемы, вызвав соответствующую функцию.

Что делает подсистема

На данном этапе реализации алгоритма подсистема делает следующее.

Допустим, к подсистеме пришел запрос на регистрацию. В параметрах этой регистрации передается структура, содержащая имя регистрируемого узла и указатель на функцию, которая должна возвращать его локальное время. Остальные поля: таймер и два последних значения локального счетчика, узел заполнять не должен.

Если что-то из обязательных полей не заполнено, то запрос будет отклонен.

Подсистема для каждого узла запускает таймер, частота срабатывания которого выбирается на основании информации, предоставленной узлом. В результате периодических срабатываний таймера мы знаем два последних значения локального времени узла, с каким системным временем каждое из них связано, и за какое системное время они произошли, что позволяет легко пересчитать очередное время узла в системное.

Реализация предельно проста и лаконична, и подчиненному узлу очень легко эту подсистему использовать.

Заключение

Отметим преимущества реализованного алгоритма:

- весьма прост в реализации и использовании
- не вызывает торможения, ускорения или скачкообразного изменения локальных часов всех зарегистрировавшихся узлов
- индивидуальный подход подсистемы к каждому зарегистрированному узлу - для каждого свой таймер со своим промежутком времени для срабатывания
- высокий уровень синхронизации между узлом и подсистемой

Из минусов следует помнить о том, что:

- уровень синхронизации непосредственно узлов друг с другом он относительно мал. Чтобы узлы могли сопоставлять свое время, им придется пересчитать свои значения в системные, и уже их сравнивать
- в данной реализации не учитывается время, затрачиваемое на обмен метками времени между узлом и подсистемой
- если подсистема выйдет из строя, то все узлы потеряют возможность пересчета своего времени

В дальнейшем планируется совершенствование алгоритма. В частности, планируется:

- возможность выбора одного из нескольких алгоритмов пересчета времени в зависимости от той информации, которую узел предоставил при регистрации, что позволит выбрать для каждого узла оптимальный алгоритм.
- учет времени, затрачиваемого на пересылку меток времени
- динамическое корректирование стратегии пересчета в каждом алгоритме в зависимости от общей динамики хода локального счетчика каждого узла. Необходимость в этом вызвана тем, что время может изменяться не по абсолютно линейному закону, а со скачками, и для того, чтобы эти скачки не портили правила пересчета, надо учитывать, как время в узле шло раньше текущего запроса на новое значение счетчика

Список литературы

[1] Precision Time Protocol on Linux.
Автор: Ken ICHIKAWA, FUJITSU LIMITED. LinuxCon Japan 2014
Файл: 1 Precision Time Protocol on Linux.pdf
Дата: 2014

[2] Computer Time Synchronization Concepts

Авторы: Martin Burnicki, Meinberg Funkuhren, Bad Pyrmont. Germany
Файл: 2 Computer Time Synchronization Concepts.pdf
Дата: 2014-04-29

[3] Hardware Assisted Precision Time Protocol. Design and case study

Авторы: Patrick Ohly, David N. Lombard, Kevin B. Stanton

Файл: 3 Hardware Assisted Precision Time Protocol. Design and case study.pdf

[4] Синхронизация времени в распределенных системах, в зависимости от выбранной модели непротиворечивости.
Автор: Ю.А. Алдунин
Файл: 4 Ю.А.Алдунин.pdf
Дата: 2008
[5] wikipedia.org/
[6]
http://embeddedpro.ucoz.ru/app_notes/distributed_computing_principles.html#Синхронизация
[7] The Accuracy of clock synchronization
Авторы: Riccardo Gusella, Stefano Zatii
Файл: 5 Berkeley Synchronization Protocol.pdf
Дата: 01.1987