1 Аннотация

В распределенных вычислительных системах иногда возникает необходимость синхронизации времени на узлах таких систем с некоторым эталонным источником времени. Сегодня существует ряд алгоритмов для решения этой задачи с различной эффективностью и точностью. В данной работе представлен обзор существующих алгоритмов, а так же подробный разбор и предложения по улучшению одного из них, реализациях этих улучшений.

Так же в рамках работы представлены измерения эффективности улучшаемого алгоритма до его модернизации и после.

2 Введение

Рассмотрим следующую систему: имеется машина с установленной Linux-based операционной системой и различными подключенными внешними устройствами. На машине запущены одна или несколько программ, которым для работы требуется, чтобы системные часы шли согласно некоторому эталонному источнику времени. Например, такой программой может быть драйвер сетевого адаптера, который должен регистрировать приходящие в систему извне пакеты данных и ставить в соответствие каждому из них некоторую метку времени, а источником могут быть часы другой системы, или одно из устройств, подключенных к рассматриваемой системе.

То есть, таким программам требуется, чтобы системные часы были *синхронизированы* с эталонным источником. Это означает, что в каждый момент времени значения часов источника и того, кто с ним синхронизируется, должны как можно меньше отличатся. Чем меньше разница, тем выше точность синхронизации.

Источников, как и программ, нуждающихся в синхронизации, может быть несколько. Для синхронизации значений часов системы может быть использован NTP (Network Time Protocol), дающий относительно неплохую точность синхронизации, порядка миллисекунд [5]. Но иногда такой точности недостаточно, и тогда помогают алгоритмы, точность которых достигает порядка наносекунд.

Рассматриваемый в данной работе алгоритм позволяет с помощью некоторого высокоточного генератора PPS (Pulse per second) сигналов узнавать о начале каждой новой секунды и сообщать об этом системе, в которой, в свою очередь, регистрируются приложения - PPS-потребители, которым требуется реагировать на эти сигналы. Каждый PPS-сигнал вызывает в системе рассылку текущего значения системных часов всем зарегистрированным потребителям.

Для получения высокой точности можно использовать какой-нибудь относительно грубый алгоритм, например NTP, в совокупности с PPS [2]. И тогда если время будет синхронизироваться с помощью основного алгоритма до хотя бы секунд, то уже внутри секунды синхронизацию обеспечит PPS.

Рассмотрим далее более конкретно, что такое метки времени, откуда берется PPS-сигнал, и что это такое, какие сложности возникают при реализации алгоритмов, работающих с PPS, а так же конкретный пример - алгоритм, использующийся в ядре Linux для синхронизации системного времени с PPS-источником.

3 Время в ядре Linux

В компьютере время представлено как значение некоторого постоянно возрастающего аппаратного счетчика, значение которого увеличивается через примерно равные промежутки времени. Счетчик не может инкрементироваться через просто равные промежутки времени, так как частота его инкрементации связана с частотой некоторого генератора, а его частота колеблется в относительно небольшом диапазоне, зависящем от условий, в которых находится генератор, например, от давления и температуры, от энергопотребления и других причин [3].

Значение этого счетчика в каждый момент времени - целое число, которое используется системой для перевода его в человеческие единицы измерения времени. Говоря о человеческих единицах времени будем здесь и далее иметь ввиду наносекунды. Для перевода значения счетчика в наносекунды требуется знать, с примерно какой частотой счетчик инкрементируется - для каждой системы устройств эта частота известна заранее.

Значение этого счетчика - это так называемое *raw monotonic time* - "сырое" значение времени. Так же в системе есть понятие *real time* - это время, получаемое из raw monotonic с помощью различных коэффициентов пересчета. Применение алгоритмов синхронизации времени - это способ улучшать эти коэффициенты для получения более точного времени. real time - это то время, которое доступно в пользовательской области операционной системы для приложений вне ядра, и которое обычно называется *системными часами*.

Когда говорят о коррекции времени, то имеют ввиду не прямое изменение значения аппаратного счетчика, а способ его преобразования в real time из raw monotonic time. То есть алгоритмы синхронизации служат для построения правил, по которым вычисляются коэффициенты для пересчета raw monotonic time.

Для инкрементации значения счетчика времени существует несколько механизмов, вот одни из них:

1. Аппаратные прерывания
2. Time stamp counter (TSC)
3. High precision event timer (HPET)

Прерывание - это сигнал, сообщающий процессору о наступлении какого-либо события, управление при возникновении прерывания передается его обработчику [6]. Прерывание может быть как аппаратным, в случае счетчика времени, так и программным, если оно инициировано специальной инструкцией в коде. То есть через примерно равные промежутки времени счетчик инкрементируется с помощью аппаратного прерывания. Но этот способ является устаревшим, поскольку обработка прерывания - это дополнительная нагрузка на систему.

Time stamp counter - специальный регистр процессора, который считает количество тактов, прошедшее с запуска компьютера. Его значение можно получить специальной инструкцией в коде.

HPET - аппаратный таймер, используемый в персональных компьютерах. Фактически, это отдельное устройство, подключенное к процессору. Метку времени можно читать с его регистров. Так же это устройство можно программировать на прерывания.

Выше была рассмотрена система, часы которой надо синхронизировать, и устройство, являющееся источником PPS сигналов. У системы есть свой счетчик времени, значение которого менять нельзя, но можно менять коэффициенты и правила, по которым считается real time. Вычислением таких коэффициентов и занимается алгоритм синхронизации, использующий PPS.

4 Pulse per second

Pulse per second (PPS) - сигнал, генерируемый некоторым высокоточным источником (PPS-source) с частотой максимально близкой к одной секунде [2]. Для рассматриваемого в данной работе алгоритма важно уметь "ловить" моменты, когда возникает PPS-сигнал, то есть получать метки времени точно в эти моменты.

PPS-источник подключается к вычислительной системе посредством контакта интерфейса последовательной линии (modem-control pin on a serial-line interface) или параллельной . Система узнает о возникновении сигнала, получая прерывание, и как можно скорее запоминает текущую метку времени. Далее эта метка становится доступна с относительно небольшой задержкой после ее получения для ПО, работающего со временем - PPS-потребителей.

PPS-потребители - это приложения, заинтересованные в обработке возникающих PPS-сигналов, которые ранее зарегистрировались в системе, как получатели PPS-сигналов. С помощью метки, полученной с PPS-сигналом, PPS-потребитель может, например, получить расхождение между источником точного времени и системными часами.

К одной системе может быть подключено несколько PPS-источников, каждый из которых генерирует собственные PPS-сигналы независимо от других. Когда очередной PPS-потребитель регистрируется на прием сигналов, он может указать, какой PPS-источник ему нужно слушать, причем слушать можно несколько источников.

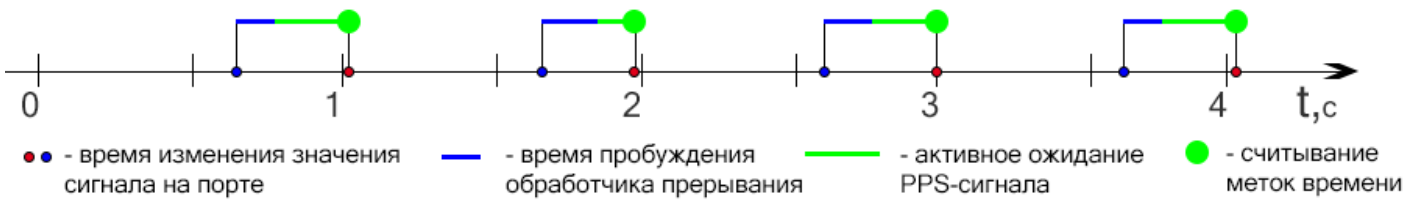
Кроме пользовательских программ в ядре есть еще один потребитель сигналов: специальная процедура ядра - *hardpps*. Эта процедура на основании получаемых сигналов пересчитывает коэффициенты, по которым monotonic time пересчитывается в real time, в результате чего изменяется системное время. Причем эта процедура может слушать единовременно только один PPS-источник. Как раз с ним рассматриваемая система и будет синхронизировать свои часы.

При реализации алгоритмов синхронизации, основанных на PPS, возникают следующие сложности:

1. Как правильно учесть время, которое пройдет с момента появления сигнала и возникновения прерывания до вызова его обработчика и запоминания текущего времени для его передачи PPS-потребителям?

2. Как использовать получаемые метки для коррекции системного времени в процедуре hardpps, описанной выше?

Рассмотрим, в чем суть первой задачи, с помощью следующего рисунка:



На рисунке изображена шкала времени, разделенная по секундам. На нем схематически изображена схема получения меток времени по PPS-сигналу.

В обработке PPS-сигналов важны не сами PPS-сигналы, а моменты времени, в которые они появляются - каждому сигналу надо как можно точнее поставить в соответствие метку времени.

Если бы прерывание возникало непосредственно в момент времени возникновения PPS-сигнала, то на пробуждение обработчика прерывания и само считывание меток уходило бы слишком много времени, причем эти временные затраты были бы непредсказуемы на каждом новом сигнале, то есть время пробуждения обработчика и считывания было бы каждый раз разным, что зависело бы от некоторых факторов, например:

- уровень энергопотребления процессора (процессор постоянно пытается его снизить). Если энергопотребление снижено, то необходимо время, чтобы "пробудить" процессор.
- если в данный момент в ядре запрещены прерывания.
- могут возникать более приоритетные прерывания.

В качестве решения используется следующий алгоритм: источник PPS-сигнала изменяет значение электрического сигнала, посылаемого на порт, немного раньше, до возникновения самого PPS-сигнала. Система, которая должна получить сигнал, вызывает прерывание в ответ на это изменение. На рисунке эти моменты обозначены красными точками на шкале. Затем обработчик прерывания просыпается - синие линии на рисунке - и переходит в режим активного ожидания PPS-сигнала - зеленые линии на рисунке. За PPS-сигнал принимается обратное изменение значения электрического сигнала на порте, - красные точки на рисунке - но в этот момент времени обработчик прерывания находился в активном ожидании этого события и может сразу считать метки времени - зеленые точки на рисунке.

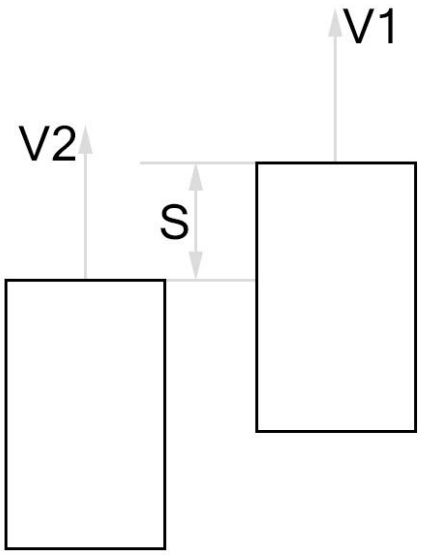
В итоге, если подготовиться к сигналу пораньше, то почти всегда метка будет считана достаточно точно. Далее рассматривается вторая из задач, описанных выше, решение которой и является рассматриваемым в работе алгоритмом.

5 Алгоритм коррекции времени

Для разбора алгоритма коррекции времени необходимо знать об алгоритме NTP, как он работает, и какие проблемы призван решить.

Network Time Protocol (NTP) - алгоритм, используемый для синхронизации часов системы с одним или несколькими удаленными серверами посредством сети [5]. Реализация NTP - это процесс-демон, работающий в пользовательском пространстве операционной системы, не в ядре. Поскольку синхронизация времени требует вмешательства ядра для коррекции коэффициентов пересчета raw monotonic time в real time, то для этого используется механизм системных вызовов.

С часами могут возникать следующие проблемы, которые и призван решать алгоритм NTP: проблема расхождения фазы часов, и частоты хода часов. Чтобы лучше понять, в чем суть этих двух расхождений, можно рассмотреть следующий пример:



Пусть есть дорога, по которой движутся два автомобиля. Пусть они так же движутся с разными скоростями. Тогда расстояние между ними - S - будет постепенно увеличиваться. Если провести аналогию с часами, то автомобили - это разные часы, их скорости - это частоты хода часов, а S - это постоянно возрастающая разница показаний этих часов.

В данном случае просто скорректировать S недостаточно - оно все равно будет снова возрастать из-за разных частот. Поэтому здесь требуется коррекция именно частот хода часов.

Что значит ошибка фазы? - пусть снова есть два автомобиля, которые движутся с одинаковыми скоростями рядом друг с другом. Пусть теперь один из автомобилей, сохраняя прежнюю скорость вращения колес, пробуксовал на

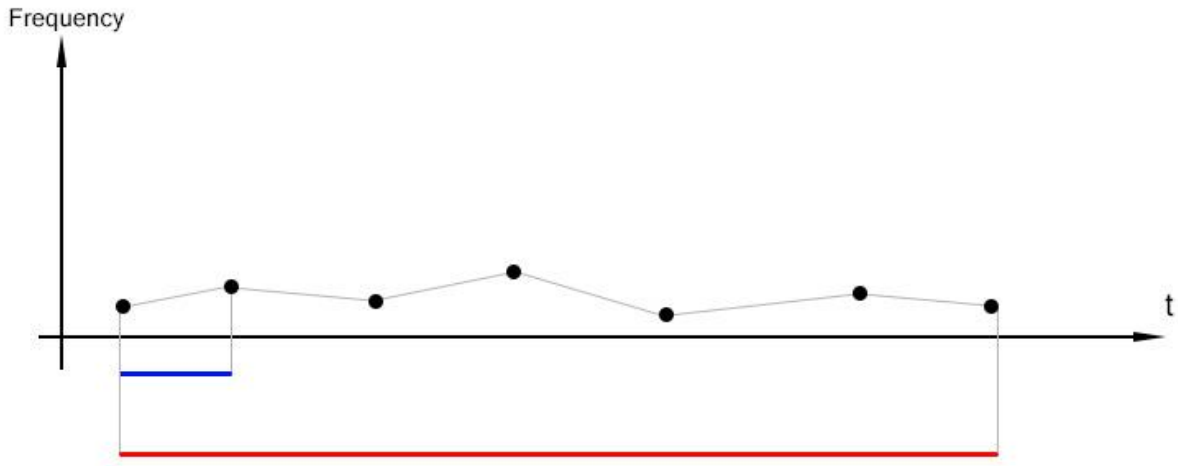
льду, немного отстав от второго автомобиля. Скорости автомобилей остались прежними, но дистанция между ними стала S. Если провести аналогию с часами, то здесь произошла ошибка фазы S. Исправлять такие ошибки фазы можно за один раз, до следующего сбоя ("пробуксовки").

Таким образом, алгоритм NTP исправляет сразу две ошибки: ошибку фазы, и ошибку частоты. Для этого используется множество коэффициентов, которые обновляются по мере работы алгоритма.

Часть этих коэффициентов обновляется специальной процедурой - *hardpps*. Эта процедура является частью ядра, а не демона NTP, работающего в пользовательском пространстве, и вызывается при каждом возникновении PPS-сигнала, чтобы обновить параметры алгоритма NTP, которые используются для вычисления коэффициентов пересчета raw monotonic time в real time.

Алгоритм коррекции времени состоит из двух самостоятельных частей: первая корректирует фазу, а вторая частоту, и называются они соответственно Phase-Locked-Loop и Frequency-Locked-Loop [5].

В то время, как фаза корректируется каждую секунду, частота корректируется реже, раз в T_c секунд. T_c - это один из параметров алгоритма, который называется *длина интервал частоты*, и он меняется по мере работы алгоритма в некотором диапазоне. Зачем для частоты нужен именно относительно длинный интервал, а не просто две соседних секунды, как с фазой?



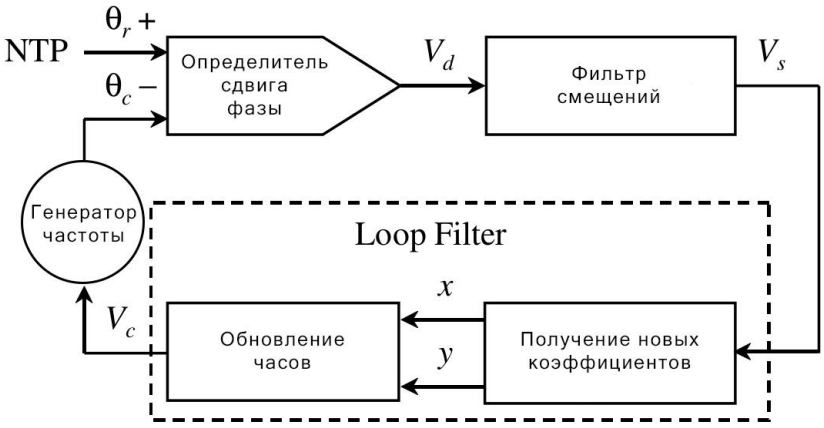
Что вообще такое частота хода часов в терминах алгоритма NTP? Это насколько единиц увеличивается raw time за одну секунду. Поскольку PPS-сигналы возникают раз в секунду, частота должна быть 1000 000 000 единиц в секунду, так как raw time имеет разрешение порядка наносекунд. Поскольку частота хода часов зависит от частоты некоторого аппаратного генератора, то она постоянно колеблется, и вычислять ее по двум соседним секундам - на рисунке это синяя полоса - может быть ненадежно. То есть для получения более среднего значения частоты используют более длинные интервалы - красная линия на рисунке выше. Если на этом интервале вычисленная частота отклоняется от эталонной, то она корректируется на возникшую разницу.

Когда частота становится менее стабильной, то интервал частоты уменьшается, так как частота нуждается в более частой корректировке, а когда частота стабильна, то обновлять ее можно относительно редко.

Что может служить причиной ошибок фазы и ошибок частоты? Ошибки фазы возникают обычно из-за некоторых программных причин, например, это могут быть различные задержки в операционной системе. А ошибки частоты возникают из-за колебаний температуры генератора частоты, перепадов напряжения. NTP умеет определять, какие ошибки в системе преобладают, и использует исправления частоты и фазы с разными весами [5].

На выходе такого комбинированного алгоритма получаются коэффициенты, которые используются для коррекции времени.

Алгоритм работает, как замкнутая система управления *(feedback control system)* следующего вида:

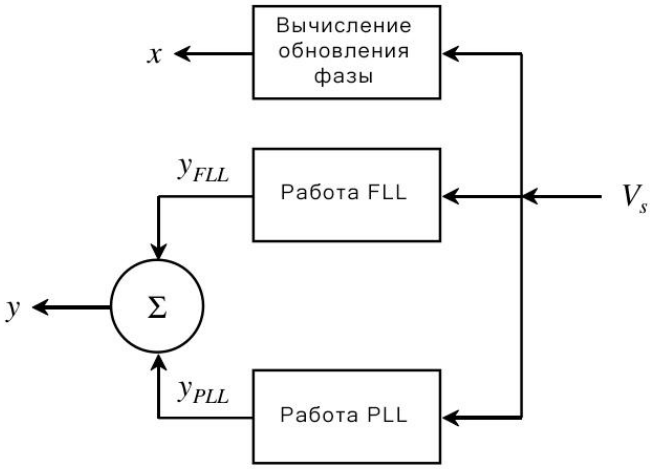


θ_r представляет эталонное время, θ_c - системное время. Сдвиг определяется как разница этих значений $V_d = \theta_r - \theta_c$.

Фильтр смещений отсеивает неправдоподобные значения V_d. Так же в фильтре NTP использует различные алгоритмы выбора, кластеризации и комбинирования данных через несколько фильтров, чтобы получить наилучшее смещение V_s. В случае использования hardpps вместо внешних серверов с эталонным временем этот фильтр просто отбрасывает заранее невозможные смещения, например, слишком большие.

Далее алгоритм вычисляет смещения частоты и фазы - *y* и *x,* которые уже используются для генерации сигнала V_c , который каждую секунду корректирует сдвиг фазы системы, используя вычисленные коэффициенты.

Как считаются *x* и *y*?



Здесь y_{PLL} и y_{FLL} - функции над V_s и *x*, они будут рассмотрены подробнее далее. *x* - называется остаточной ошибкой фазы, и используется для ежесекундной коррекции фазы.

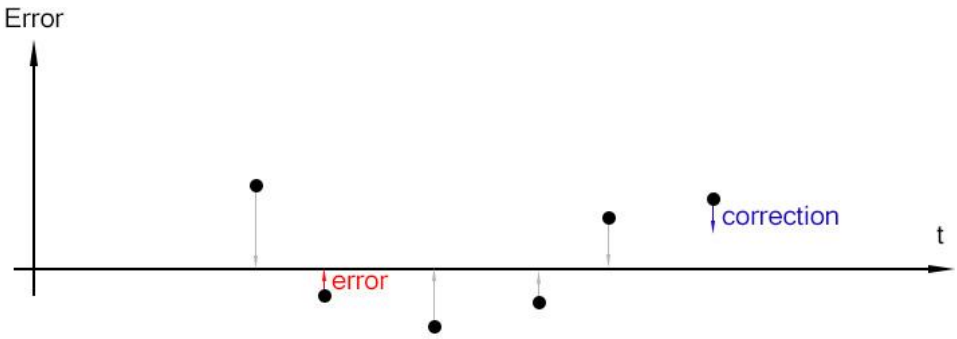
Обобщая работу NTP - цель состоит в том, чтобы как можно сильнее уменьшить V_s к следующему шагу работы алгоритма.

Рассмотрим роль процедуры hardpps в этом алгоритме. *hardpps* - так называется специальная процедура ядра, служащая для синхронизации системных часов с некоторым PPS-источником, и реализована она как C функция в коде ядра. При этом hardpps является частью реализации алгоритма NTP, но находящейся именно в ядре. hardpps вызывается при каждом возникновении сигнала и работает с двумя аргументами - метками времени real time и raw monotonic time, запоминаемыми в момент получения сигнала.

На каждом вызове hardpps корректирует параметр алгоритма NTP - смещение фазы времени. Для этого вычисляется количество наносекунд в

текущей секунде метки real time - обозначим как *nsecs.* Алгоритм во время работы использует и обновляет *кольцо* последних значений nsecs. На каждой итерации из кольца выбирается некоторое наилучшее значение - *correction*.

Алгоритм выбора correction может быть реализован по разному, например, в оригинальном ядре используется выбор просто последнего положенного в кольцо значения. Так же есть более сложные алгоритмы: например, выбор медианы из значений кольца, или среднего значения, но у медианного способа сходимость слишком медленна, а среднее значение неустойчиво к сильным отклонениям. В качестве альтернативы есть так называемый "смарт фильтр".



Пусть черные точки - это последние отклонения, записанные в кольцо. Смарт фильтр для вычисления correction использует минимальное из абсолютных значений кольца - здесь это error, а в качестве знака - противоположный от последнего значения кольца, и в итоге получается correction. Смарт фильтр обеспечивает хорошую сходимость, и не чувствителен к разовым сильным отклонениям - выбросам.

Далее используется фильтр, который проверяет, насколько ошибка nsecs изменилась с прошлой итерации. Если ошибка стала слишком велика, то корректировка времени не происходит. Иначе время обновляется на значение correction, но не сразу - это исправление "распределяется" по секунде, чтобы не было резкого скачка. В оригинальном ядре, например, в течение секунды тысячу раз применяется по одной тысячной от correction.

Коррекция параметров NTP, отвечающих за частоту, происходит следующим образом:

1) Вычисляется скорость хода часов, как отношение изменения raw time с последнего обновления параметров к длине этого интервала обновления.
2) Полученное отношение выставляется как обновление частоты

Как конкретно вычисляются *x, y,* y_{PLL} и y_{FLL}, и как выглядит и меняется T_c?

T_c - интервал частоты, рассмотренный ранее, имеет вид 2^τ, где τ уже меняется в некотором интервале. Соответственно, когда частота становится нестабильна, то τ уменьшается на единицу вплоть до некоторой нижней границы, а когда частота стабильна, то τ увеличивается.

Сложнее вычисляются формулы других параметров алгоритма:

$$y_{PLL} = \frac{V_s \mu}{(64T_c)^2}, y_{FLL} = \frac{V_s - x}{8\mu}$$

Стоит отметить, что на каждом обновлении y_{FLL} считается в предположении, что x на следующем обновлении будет нулем. А если это не так, то V_s надо уменьшить на эту величину, чтобы исправить только влияние частоты.

$$y = y + y_{PLL} + y_{FLL}$$

Когда заканчивается очередной интервал частоты, то x принимается равным V_s.

Каждую секунду, то есть в течение очередного интервала частоты, процесс коррекции времени вычисляет обновление для фазы $z = \frac{x}{16T_c}$ и новый $x = x - z$. Величина z используется ядром для коррекции фазы. Так продолжается до следующего обновления, когда x и y будут пересчитаны заново.

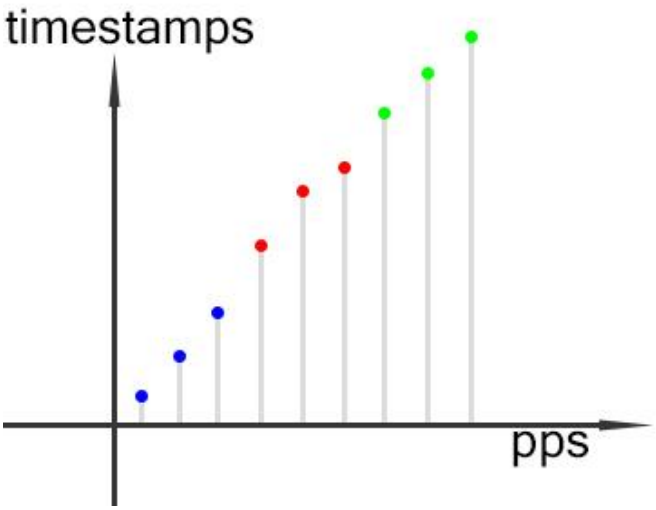
Далее будет показано, что описанный выше алгоритм нуждается у улучшении, и как это улучшение можно произвести.

6 Улучшение алгоритма коррекции времени

Здесь рассматривается улучшение той части алгоритма, что ответственна за корректировку частоты, так как это наиболее "уязвимая" часть алгоритма, и так как для улучшения корректировки фазы уже предложен ряд фильтров: медианный; смарт, описанный выше; среднее арифметическое и т.д.

Как было описано в предыдущем разделе, корректировка частоты происходит не регулярно, а раз в T_c секунд, причем T_c меняется по мере

работы алгоритма. То есть все время работы алгоритма можно разбить на интервалы, которые используются для вычисления корректировки частоты.



На рисунке изображен пример разбиения времени работы алгоритма корректировки частоты на интервалы. Точками обозначены метки времени, снятые в момент возникновения PPS-сигнала. Разными цветами обозначены разные интервалы частоты.

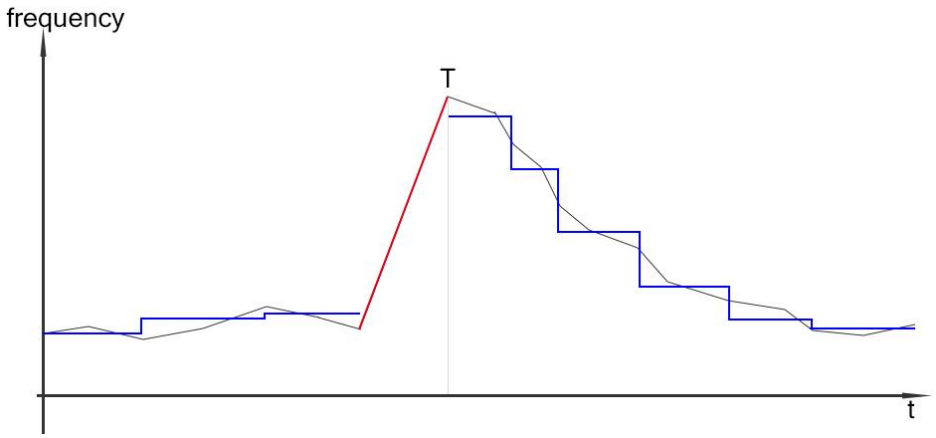
Рисунок схематический и на самом деле интервалы частоты могут быть разной длины. Интервал может закончится одним из двух способов: либо просто выделенное на него время истекло, либо произошел выброс - сильное отклонение метки времени от ожидаемого значения.

Выбор длины нового интервала происходит следующим образом: если последние несколько интервалов завершились нормально, то есть не в результате выброса, а так же новая частота не отклоняется от старой более, чем на некоторое значение, то длина возрастает в два раза. Если же что-то из этого не выполнено, то длина уменьшается в два раза. Длина интервала частоты ограничена снизу 4 секундами, и сверху - 256-ю.

Такой алгоритм имеет ряд недостатков:

1) Если текущий интервал достаточно длинный и закончится не скоро, а частота вдруг изменится, то ее корректировка не будет происходить до конца интервала и качество синхронизации ухудшится. Например, если длина интервала 256 секунд, и до его конца еще 120 секунд, а частота изменилась сейчас, то еще две минуты синхронизация будет терять точность.
2) Даже если интервал относительно короткий (4 или 8 секунд), алгоритм все равно будет плохо справляться с монотонным изменением частоты, если она меняется раз в 2-3 секунды.

Обе проблемы свидетельствуют о некоторой "неповоротливости" алгоритма обновления частоты, которая может быть критична для ряда приложений. Рассмотрим подробнее суть второй проблемы.



На рисунке выше изображена работа алгоритма на некоторых данных. На ось ординат отображаются значения частоты, высчитанной на каждом интервале частоты, а на ось абсцисс - время работы алгоритма. Серым обозначена реальная частота, а синим - полученная с помощью алгоритма. До момента времени T эталонный источник частоты работал относительно стабильно. Но пусть в момент времени T эталонный источник был перезагружен, или, например, перегрелся, в результате чего частота резко увеличилась и сразу начала медленно спадать обратно.

В момент резкого выброса T интервал частоты сбрасывается (обозначено красным), а затем алгоритм возобновляет работу. Причем частота может монотонно уменьшаться так, что, это не приведет к уменьшению длины интервала частоты, в результате чего просчитанная частота может сильно отличаться от реальной, и будет обновляться слишком редко. Таким образом возникает задача доработать механизм корректировки длины интервала частоты - правильно выбранная стратегия обновления длины интервала поможет решить обе упомянутые выше проблемы. Далее представлено два варианта доработки алгоритма обновления частоты, а так же их показатели эффективности.

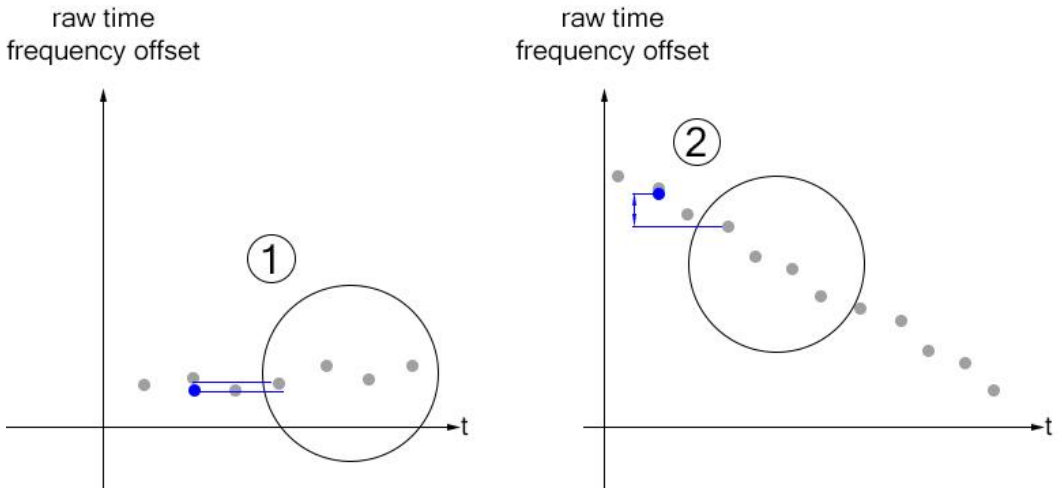
7 Анализ последних значений частоты

Значение частоты - это на сколько единиц увеличилось raw time за единицу времени. Эталонной частотой считается 1000 000 000 единиц в секунду. Рассмотрим способ улучшения алгоритма коррекции частоты на основе анализа последних отклонений от эталонной частоты.

На каждом возникновении PPS-сигнала предлагается запоминать, на сколько увеличилась метка raw time с предыдущего шага. Отклонение этой величины от эталонной запоминается алгоритмом. Достаточно единовременно хранить несколько последних отклонений, например, 4.

Затем происходит проверка, насколько текущая коррекция частоты, полученная из предыдущего интервала частоты, далеко от последних известных коррекций. Если минимальная абсолютная разница между текущей коррекцией частоты и запомненными значениями выше некоторого значения, то это означает, что частота меняется быстрее, чем обновляется ее коррекция. В таком случае интервал частоты перезапускается, а результаты предыдущего применяются. Так же уменьшается длина интервала частоты, так как большое отклонение от последних значений свидетельствует о неустойчивости частоты.

На следующем рисунке представлен пример работы алгоритма с описанной модификацией.



Здесь серые точки - смещения частоты относительно эталонной на момент возникновения очередного PPS-сигнала. Синими точками обозначены смещения частоты, полученные из очередного интервала частоты, а в круги занесены смещения, которые в текущий момент времени запомнены алгоритмом.

В первом случае текущее смещение частоты слабо отклоняется от кольца, поэтому нет необходимости перезапускать интервал. Во втором случае частота монотонно убывает, например, после резкого скачка вверх. Затем через некоторое время оказывается, что текущая коррекция начинает отставать от кольца последних смещений - это повод запустить новый интервал, применив результаты текущего.

Во втором случае длина интервала до скачка вверх могла быть 256 и тогда потеря точности будет очень велика, если не обновлять так долго на монотонно убывающей частоте.

Таким образом, с помощью описанного алгоритма можно быстро реагировать на многократные изменения частоты. Но такой метод не улучшит ситуацию с разовым изменением частоты на длинном интервале. Далее описывается второй из способов улучшения алгоритма коррекции частоты, который способен справится с этой проблемой.

8 Инверсии в кольце смещений фазы

Инверсия последовательности чисел - это такая пара чисел x и y в ней, что x расположен в ней раньше y, но при этом x > y [7]. Инверсии же в кольце смещений фазы определим по аналогии, но с некоторой правкой - не x > y, а |x| > |y|, то есть будем сравнивать по абсолютным значениям.

При таком определении большое количество инверсий свидетельствует о том, что смещения фазы сходятся к нулю, то есть алгоритм синхронизации времени сходится. Если же инверсий становится мало, то это означает, что смещения фазы возрастают по модулю, то есть время расходится с эталонным источником.

Предлагаемое улучшение заключается в проверке на каждом PPS-сигнале количества инверсий в кольце смещений фазы, и если оно меньше некоторого порога, то это означает, что коррекция фазы не справляется с задачей, и пора скорректировать частоту. В таком случае запускается новый интервал частоты с меньшей длиной, а так же применяются результаты текущего интервала.

Такой способ улучшения алгоритма корректировки частоты позволяет вовремя среагировать на ситуацию, когда в течение длинного интервала частоты она меняется и начинает нарастать ошибка фазы, но его сложность

имеет порядок $O(N^2)$, где N - размер кольца, в отличие от предыдущего способа улучшения, сложность которого лишь $O(N)$.

9 Эффективность работы алгоритма

Для анализа был выбран ряд характеристик:

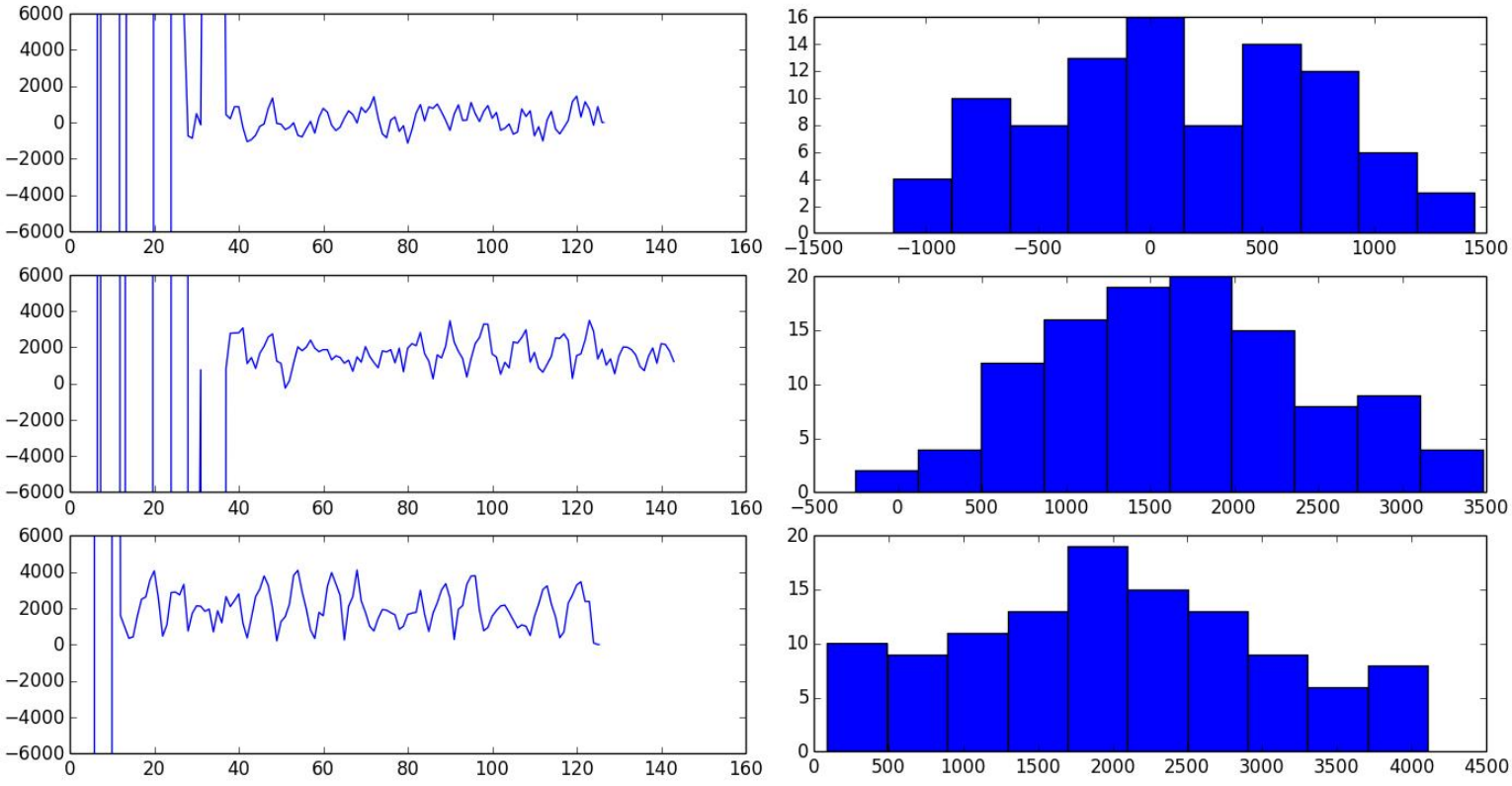
- время сходимости алгоритма
- математическое ожидание ошибки фазы
- среднеквадратичное отклонение ошибки фазы

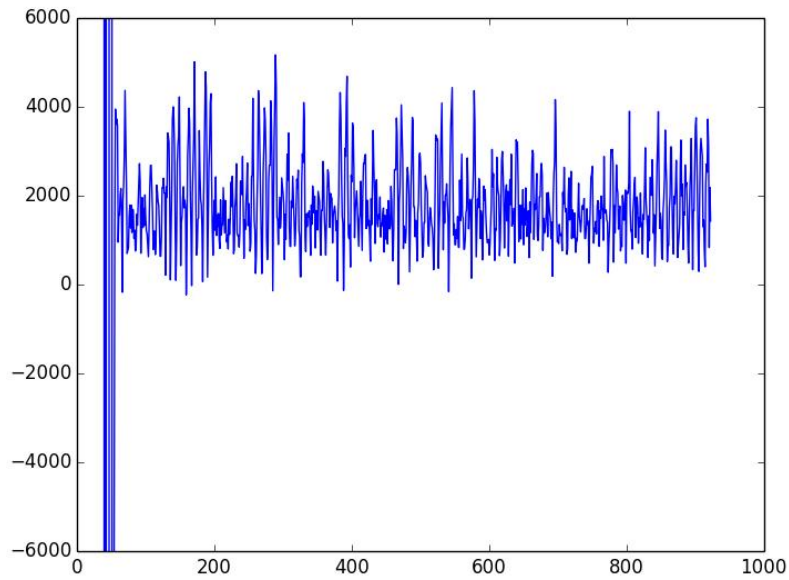
Так же по отклонениям фаз построены гистограммы. Для анализа выбраны именно ошибки фазы, так как ошибки частоты так же на них отражаются, но не наоборот.

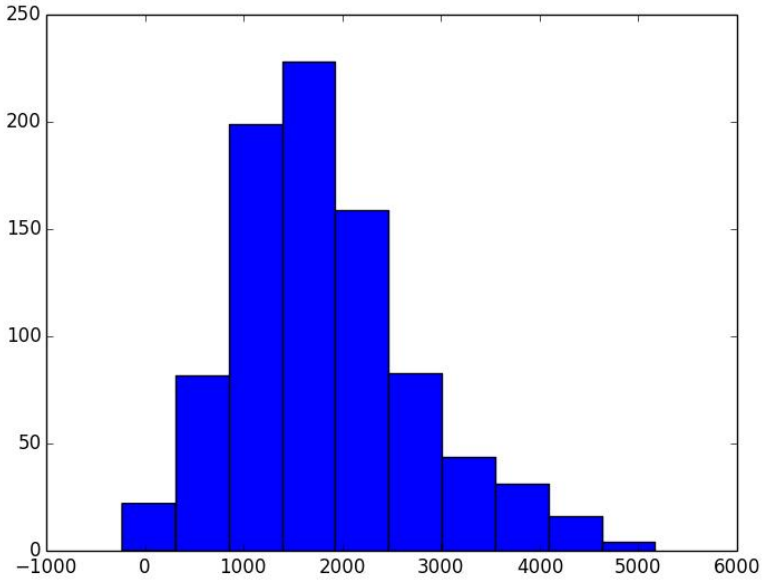
В качестве данных для анализа были использованы записи реальной работы алгоритма: три записи по две минуты и по одной записи 15 минут на каждый фильтр до улучшения алгоритма коррекции частоты и после. В конце главы приведен анализ полученных результатов.

На графиках слева показано изменение смещений фазы в зависимости от времени. На графиках справа - соответствующие гистограммы смещений фазы.

Смарт фильтр







Временем сходимости здесь называется время с начала работы алгоритма, спустя которое первый раз на трех PPS-сигналах подряд смещение фазы было менее 2000 наносекунд.

Номер графика	Длина записи	Математическое ожидание	Среднеквадратичное отклонение	Время сходимости
1	126 сек.	731 нс.	3235 нс.	28 сек.
2	144 сек.	1711 нс.	564 нс.	42 сек.
3	125 сек.	2016 нс.	626 нс.	12 сек.
4	923 сек.	1829 нс.	663 нс.	60 сек.

Без фильтрации

Фильтр среднего арифметического

10 Заключение

11 Список литературы

1. Cochran R., Marinescu C., Riesch C. Synchronizing the Linux system time to a PTP hardware clock //Precision Clock Synchronization for Measurement Control and Communication (ISPCS), 2011 International IEEE Symposium on. – IEEE, 2011. – С. 87-92.
2. Mogul J. C. et al. Pulse-per-second api for unix-like operating systems, version 1.0 //Pulse. – 2000.
3. Gleixner T., Niehaus D. Hrtimers and beyond: Transforming the linux time subsystems //Proceedings of the Linux symposium. – 2006. – Т. 1. – С. 333-346.
4. Köker K., Hielscher K. S., German R. A Low-Cost High Precision Time Measurement Infrastructure for Embedded Mobile Systems //Robot Motion and Control 2007. – Springer London, 2007. – С. 445-452.
5. Mills D. L. Internet time synchronization: the network time protocol //Communications, IEEE Transactions on. – 1991. – Т. 39. – №. 10. – С. 1482-1493.
6. Rubini A., Corbet J. Linux device drivers. – " O'Reilly Media, Inc.", 2001.
7. Pemmaraju S., Skiena S. S. Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica®. – Cambridge university press, 2003.