УДК 62-501.72:681.326.7 DOI: 10.18698/2308-6033-2021-4-2074

Синхронизация часов в распределенных многомашинных вычислительных системах. Часть I

© И.В. Ашарина

АО «НИИ «Субмикрон», Москва, Зеленоград, 124460, Россия

Обоснована необходимость синхронизации часов в распределенных многомашинных вычислительных системах. Приведены основные определения, связанные с понятием синхронизации часов. Классифицируются методы синхронизации часов. Увеличение сроков активного существования сбое- и отказоустойчивых распределенных многомашинных вычислительных систем ответственного применения является одной из наиболее актуальных задач при современном уровне развития техники. Особенно это касается необслуживаемых распределенных многомашинных вычислительных систем космического применения. Разработка таких систем должна начинаться с построения моделей неисправностей и самоуправляемой деградации, обеспечивающих, во-первых, сбое- и отказоустойчивость и, во-вторых, максимальную живучесть, что возможно лишь при наличии средств синхронизации часов в таких системах. Все действия, связанные с обеспечением синхронизации любых распределенных многомашинных вычислительных систем, начинаются с построения концепции синхронизации бортовых функций, которая основывается на генерации бортового времени и включает в себя синхронизацию бортового как программного обеспечения, так и оборудования, требующего синхронизации часов или информации о ходе времени. Основными элементами этой концепции являются модуль тактового генератора процессора, часы бортового программного обеспечения, атомные навигационные часы. В первой части работы приводятся основные определения, рассматриваются методы и алгоритмы, связанные с процессом синхронизации часов. Вторая часть посвящена синхронизации в системах с византийскими неисправностями и в многокластерных и многокомплексных (многозадачных) системах. Рассмотрены современные технологии, обеспечивающие процесс синхронизации в таких системах.

Ключевые слова: *распределенная многомашинная вычислительная система, сбое- и отказоустойчивость, синхронизация часов, динамическая избыточность, враждебная (византийская) неисправность*

Введение. Широкое внедрение распределенных многомашинных вычислительных и информационно-управляющих систем (РМВС, РМИУС) сетевой структуры для автоматизации процессов управления сложными организационно-техническими комплексами ответственного применения делает чрезвычайно актуальной проблему обеспечения сбое- и отказоустойчивости таких систем.

Цель настоящей статьи — определение подходов к решению проблемы увеличения сроков активного существования сбое- и отказоустойчивых РМВС ответственного применения.

Для достижения поставленной цели решается ряд следующих задач:

– анализ необходимых условий, при которых РМВС будет гарантированно сбое- и отказоустойчивой;

– определение возможности выбора оптимальных методов синхронизации для конкретной архитектуры и задач РМВС;

– анализ нештатных ситуаций, которые способны создать проблемы при синхронизации.

В исходной системе (сети) известной структуры необходимо организовать сбое- и отказоустойчивое параллельное выполнение поступающих извне заданий, каждое из которых представляет собой параллельное исполнение совокупности взаимодействующих между собой задач. Для каждого исполняемого системой действия (решения задачи или межзадачного взаимодействия) задан требуемый уровень сбое- и отказоустойчивости его исполнения, обеспечиваемый за счет необходимой его репликации, т. е. репликационного параллельного выполнения задачи в заданной совокупности вычислителей (цифровых вычислительных машин — ЦВМ), составляющих комплекс этой задачи, с обменом результатами между вычислителями комплекса и выбором из них правильного. Для межзадачного взаимодействия требуемый уровень достигается за счет введения необходимой избыточности среды его осуществления.

Наиболее перспективным способом обеспечения сбое- и отказоустойчивого выполнения целевой задачи в РМВС является репликация этой задачи с использованием динамической избыточности. Динамическая избыточность обеспечивает системное самообнаружение проявившихся неисправностей, их самоидентификацию по месту проявления и типу (сбой, программный сбой, отказ), самовосстановление целевой работы в сбившихся ЦВМ, самореконфигурацию системы, состоящую в изоляции отказавших ЦВМ и включения вместо них и втягивания в целевую работу системы запасных ЦВМ (при их наличии) либо осуществления управляемой самодеградации системы с переходом к безопасному останову при возникновении недопустимых неисправностей или исчерпании ресурсов. Ключевым механизмом такой организации вычислений является взаимное информационное согласование (ВИС), обеспечивающее во всех исправных ЦВМ системы координацию различающейся согласуемой информации разных ЦВМ системы таким образом: во всех исправных ЦВМ общая одинаковая согласованная информация совпадает даже при возникновении враждебных (византийских) неисправностей ЦВМ допустимой кратности с согласуемой информацией каждой исправной ЦВМ. Эта информация во всех исправных ЦВМ системы составляет основу согласованных и синхронизированных действий всех исправных ЦВМ системы по обеспечению ее сбое- и отказоустойчивости.

Задача увеличения сроков активного существования сбое- и отказоустойчивых РМВС ответственного применения может быть решена

перераспределением вычислительных ресурсов, существенным моментом которого является взаимообмен сообщениями между различными ЦВМ системы только при обеспечении следующих условий:

– работа различных ЦВМ системы должна быть в определенной степени синхронизирована. Предположение относительно необходимой синхронности РМВС состоит в следующем. РМВС считается синхронизированной, если существуют некоторые константы, ограничивающие время передачи сообщений между любыми двумя ЦВМ и соизмеряющие скорости обработки информации со стороны различных ЦВМ системы [1];

– ЦВМ-получатель сообщения должна быть способна определить ЦВМ-отправителя этого сообщения;

– должна обеспечиваться согласованность действий исправных ЦВМ системы, основанная на принятии ими одинаковых решений, как в исправном состоянии системы, так и в присутствии неисправностей допустимого типа.

Механизмы, обеспечивающие эти условия, приведены в [2] и названы базовыми механизмами организации сбое- и отказоустойчивости в РМВС.

В настоящей статье анализируются вопросы синхронизации в РМВС как необходимого условия обеспечения в них сбое- и отказоустойчивости. Все начинается с построения концепции синхронизации бортовых функций, основанной на генерации бортового времени, которая включает в себя синхронизацию бортового программного обеспечения (ПО) и бортового оборудования, требующего синхронизации часов или информации о ходе времени. Основными элементами этой концепции являются модуль тактового генератора процессора, часы бортового ПО, глобальные навигационные часы ГЛОНАСС, GPS или Galileo [3]. Доступность информации о бортовом времени важна как в случае выполнения целевых функций, так и при взаимодействии отдельных ЦВМ в РМВС, установленной на большом космическом аппарате (КА), а также при организации работы группировки малых КА (рой) или даже при совместной работе роботов-манипуляторов.

Основные определения и предположения. Рассмотрим вопросы, связанные с синхронизацией часов в сбое- и отказоустойчивых РМВС ответственного применения.

Вопросы синхронизации решаются очень давно. Проблема синхронизации часов в распределенной системе впервые была исследована Л. Лэмпортом [4] в 1978 г. Он отмечал, что концепция времени лежит в основе мышления, она основана на более общей концепции порядка, в котором происходят события. Однако эту концепцию необходимо тщательно проанализировать при рассмотрении событий в распределенной системе.

Распределенная система, с точки зрения синхронизации часов, состоит из набора процессоров, которые пространственно разделены, обмениваются сообщениями и не имеют доступа к глобальным часам. Система является распределенной, если задержка передачи сообщения не пренебрежимо мала по сравнению со временем между событиями в одном процессе. В распределенной системе иногда невозможно сказать, что одно из двух событий произошло первым. Отдельный компьютер также можно рассматривать как распределенную систему, в которой центральный блок управления, блоки памяти и каналы ввода–вывода являются отдельными процессами. Все большее количество распределенных приложений, например, приложения для управления процессами, приложения для обработки транзакций или протоколы связи, требуют наличия синхронизированных часов, чтобы иметь примерно одинаковое представление о времени [4, 5].

Время в этом контексте означает либо приближение реального времени, либо просто целочисленный счетчик. Алгоритмы, обеспечивающие общее знание о текущем системном времени физически распределенными процессорами, называются алгоритмами синхронизации часов [5].

Алгоритмы синхронизации часов могут использоваться для синхронизации часов по отношению к внешней временной привязке и/или для синхронизации часов между собой [5]. В первом случае, называемом внешней синхронизацией часов, источник часов показывает реальное время, и цель всех часов — быть как можно ближе к этому источнику времени. Во втором случае, называемом внутренней синхронизацией часов, реальное время недоступно внутри системы, и цель состоит в том, чтобы минимизировать максимальную разницу между любыми двумя часами [5].

Согласно [6], локальные аппаратные часы фиксируют сбой синхронизации, если они не ограничены небольшими постоянными величинами β (рассогласованием) и τ_v (скоростью дрейфа часов).

Локальные аппаратные часы подвержены враждебной (византийской) неисправности [7] в том случае, если они выдают неточную, несвоевременную или противоречивую информацию. Это определение включает в себя понятие «двуликих» часов (dual-faced clocks), которые могут выдавать разные значения времени для разных процессоров.

Часы реального времени (real-time clock, RTC) являются физическим устройством, они испытывают на себе влияние физических факторов:

– кратковременных (помехи);

– умеренных по времени (колебания параметров окружающей среды, прежде всего температуры);

– долговременных (старение).

Влияние этих факторов сказывается на главном счетном механизме — кварцевом генераторе, замедляя или ускоряя производимую им частоту. Следовательно, можно выделить такие важные параметры кварцевого резонатора [8]:

– рабочий диапазон температур;

– отклонение частоты при нормальной температуре (статическая составляющая дрейфа; дрейф — смещение, сдвиг, уход, снос; перемещение, отклонение [9]);

– температурная стабильность (динамическая составляющая дрейфа), старение;

– нагрузочная емкость.

Системы управления жесткого реального времени предъявляют наиболее жесткие требования к скорости достижения синхронизации. Для этого широко используется стандарт IEEE 1588, который позволяет поочередно синхронизовать часы в устройствах распределенной системы, после чего синхронные часы используются для синхронизации взаимодействия устройств системы [10, 11].

Методы синхронизации. Методы синхронизации часов основаны на программном или аппаратном подходе. Программный подход — наиболее гибкий и экономичный, но вызывающий необходимость в обмене дополнительными сообщениями, что повышает информационную избыточность процесса, причем эта избыточность обусловлена именно процессом синхронизации.

Аппаратный подход требует специального оборудования в каждом узле сети, что позволяет достичь жесткой синхронизации при минимальных временных и информационных затратах. Однако стоимость дополнительного оборудования делает этот подход неприемлемым в больших распределенных системах, кроме случаев систем ответственного применения с необходимостью жесткой синхронизации.

Поиски компромиссов часто приводят к появлению гибридных методов. Таким образом, появились схемы программной синхронизации с аппаратной поддержкой, которые обеспечивают баланс между требованиями к аппаратному и программному обеспечению.

Сравним существующие алгоритмы отказоустойчивой синхронизации часов.

В [4] введены следующие важные предпосылки. Частичное упорядочение, определяемое отношением «произошло до» (обозначается $\rightarrow$): «отношение $\rightarrow$ на множестве событий системы — это наименьшее отношение, удовлетворяющее трем следующим условиям:

1) если a и b являются событиями в одном и том же процессе и a предшествует b, то $a \rightarrow b$;

2) если a — отправка сообщения одним процессом, b — получение того же сообщения другим процессом, то $a \rightarrow b$;

3) если $a \to b$ и $b \to c$, то $a \to c$. Два различных события *a* и *b* называются одновременными, если $a \quad b$ и $b \quad a$.

В [4] приведен распределенный алгоритм для его расширения до согласованного полного упорядочения всех событий. Этот алгоритм может предоставить полезный механизм для реализации распределенной системы. Неожиданное, аномальное поведение может возникнуть, если порядок, полученный с помощью этого алгоритма, отличается от того, который воспринимается пользователем. Этого можно избежать, введя реальные, физические часы.

Ввод в систему часов начинается с абстрактной точки зрения, в которой часы — это способ присвоения числа событию, где число рассматривается как время, в которое произошло событие.

Пусть в системе выполняется некоторое количество *n* процессов:

$$\Pi = \{\pi_1, \pi_2, \ldots, \pi_n\},$$

для которых возможно возникновение некоторых событий *a*, *b*, *c*, *d*, … . Часы C_i определяются для каждого процесса π_i как функция, которая присваивает номер $C_i\langle a\rangle$ любому событию *a* в этом процессе. Вся система часов представлена функцией *C*, она присваивает любому событию *b* число $C\langle b\rangle$, где $C\langle b\rangle = C_j\langle b\rangle$, если *b* — событие в процессе π_j. Здесь не делается никаких предположений об отношении значения $C_i\langle a\rangle$ к физическому времени, поскольку речь идет о часах C_i как о логических, а не как о физических часах. Они могут быть реализованы посредством счетчиков без реального механизма синхронизации.

Далее в пространственно-временную картину введена физическая временная координата. Пусть $C_i(t)$ обозначает отсчет часов C_i в физический момент времени *t*. Для математического удобства предполагается, что часы работают непрерывно, а не дискретно (дискретные часы можно рассматривать как непрерывные, в которых существует ошибка до «тика» при их считывании), точнее, предполагается, что $C_i(t)$ является непрерывной дифференцируемой функцией *t* за исключением изолированных скачкообразных разрывов, где часы сбрасываются. Тогда $dC_i(t)/dt$ представляет скорость, с которой часы работают в момент времени *t*.

Для того чтобы часы C_i были настоящими физическими часами, они должны работать примерно с правильной скоростью *l*, т. е. должно выполняться $dC_i(t)/dt \approx l$ для всех *t*. Точнее, должно выполняться следующее условие:

РС1. Существует такая константа $k \ll l$, что для всех i выполняется $|dC_i(t)/dt - l| < k$. Для типичных кварцевых управляемых часов $k \le 10^{-6}$.

Недостаточно, чтобы часы по отдельности работали примерно с правильной скоростью. Они должны быть синхронизированы так, чтобы $C_i(t) \approx C_j(t)$ для всех i, j и t, т. е. должна существовать достаточно малая константа ε, чтобы выполнялось условие:

РС2. Для всех i, j: $|C_i(t) - C_j(t)| < \varepsilon$.

При работе с распределенной системой важно понимать, что порядок, в котором происходят события, является лишь частичным. Автор [4] считает, что это положение необходимо для понимания любой многопроцессорной системы и что оно должно помочь понять основные проблемы многопроцессорной обработки независимо от механизмов, используемых для их решения.

Работа [4] определяет основные принципы синхронизации в РМВС, однако рассматриваются в ней только системы, не имеющие неисправностей.

Однако уже в 1985 г. авторы [12] разработали три отказоустойчивых алгоритма синхронизации в распределенной многопроцессорной системе, где каждый процесс имеет свои собственные часы. Эти алгоритмы работают при наличии произвольных сбоев часов или процессов, включая «двуликие» часы. Для двух из предложенных алгоритмов допустимо, чтобы менее одной трети процессов были неисправны. Третий алгоритм работает, если менее половины процессов неисправны, но требует цифровых подписей.

В [12] утверждается, что в отказоустойчивой многопроцессорной системе часто бывает необходимо, чтобы отдельные процессы поддерживали тактовые импульсы, которые синхронизированы друг с другом [13–15]. Поскольку физические часы не поддерживают точное время и могут дрейфовать относительно друг друга, часы необходимо периодически повторно синхронизировать. Такая система нуждается в алгоритме синхронизации часов, который работает, несмотря на неправильное поведение некоторых процессов и часов. При ограниченных типах допустимых отказов отказоустойчивые алгоритмы синхронизации построить достаточно просто. Однако алгоритмы, которые могут обрабатывать произвольные отказы, в частности отказы, которые могут привести к «двуликому» поведению часов, представляют собой достаточно сложную и трудоемкую задачу.

Алгоритмы, описанные в [12], работают при наличии любых неисправностей, включая вредоносные, «двуликие» часы. Первый называется интерактивным алгоритмом сходимости. В сети, состоящей как минимум из $(3m + 1)$ процессов, он может обработать до m

ошибок. Его название происходит от того факта, что алгоритм заставляет правильно работающие часы сходиться, но точность, с которой они могут быть синхронизированы, зависит от того, насколько далеко они могут дрейфовать перед повторной синхронизацией.

Следующие два алгоритма упоминаются как интерактивные алгоритмы согласованности вследствие того, что исправные процессы получают взаимно согласованные значения всех часов. Точность, с которой часы могут быть синхронизированы, зависит только от точности, с которой процессы могут считывать часы друг друга, и от того, насколько сильно они могут дрейфовать во время процедуры синхронизации. Они получены из двух основных интерактивных алгоритмов согласованности, представленных в [7]. Первый требует как минимум $(3m + 1)$ процессов для обработки до m ошибок. Второй алгоритм предполагает специальный метод чтения часов, требующий использования неподделываемых цифровых подписей, для обработки до m ошибок с помощью всего $(2m + 1)$ процессов. Последний алгоритм, по-видимому, имеет небольшую практическую ценность, поскольку авторы [16] разработали более эффективный алгоритм, основанный на том же методе считывания часов, однако он заслуживает упоминания как предшественник алгоритма из [16].

Впоследствии авторы [17] доказали, что требуется $(3m + 1)$ процессов, чтобы обеспечить синхронизацию часов при наличии m ошибок, если цифровые подписи не используются.

Алгоритмы из [12] требуют надежной, полносвязной сети связи и обрабатывают лишь произвольные ошибки процесса. Их работа ориентируется на раунды, с определенной периодичностью выполняя повторную синхронизацию, чтобы исправить дрейф часов. В каждом раунде каждый процесс получает значение для часов каждого из других процессов и устанавливает свои часы на среднее из тех значений, которые не слишком отличаются от его собственных.

Алгоритмы временной синхронизации классифицируются в соответствии с их внутренней структурой и тремя базовыми составляющими [5]:

1) блок обнаружения событий ресинхронизации, задача этого блока — определить момент, в который процессоры должны повторно синхронизировать свои часы;

2) блок оценки удаленных часов, задача этого блока — оценить значения удаленных часов;

3) блок коррекции часов, предназначение этого блока — регулировка каждых логических часов в соответствии с результатом, полученным блоком оценки удаленных часов.

Каждый из блоков обслуживается ориентированными на него алгоритмами.

Коммуникационная сеть, состоящая из множества $P = \{p_1, p_2, \ldots, p_n\}$ процессоров, соединенных между собой, может иметь различные характеристики (широковещательная или двухточечная, полносвязная (т. е. такая, моделью которой является полный граф) или неполносвязная). С каждым процессором $p_i \in P$ связаны локальные аппаратные часы C_{p_i}, обычно состоящие из генератора и регистрового счетчика, который увеличивается или уменьшается на такты («тики») генератора. Хотя часы дискретны, все алгоритмы предполагают, что часы работают непрерывно, т. е. C_{p_i} является непрерывной функцией на некотором интервале реального времени. Хотя аппаратные часы могут отклоняться от реального времени вследствие объективных причин, обычно предполагается, что их дрейф находится в пределах небольшой Δ-окрестности значений реального времени.

Для очень малой известной константы ρ > 0 (называемой скоростью дрейфа) аппаратные часы $C_{p_i}(t)$ определены как ρ-ограниченные при условии, что для реального времени *t* [5]:

$$\frac{1}{(1+\rho)} \le \frac{dC_{p_i}(t)}{dt} \le (1+\rho),$$

где $C_{p_i}(t)$ — значение аппаратных часов процессора p_i в реальном времени *t*.

Основная идея алгоритмов программной синхронизации [18] заключается в том, что каждый узел системы имеет логические часы, которые обеспечивают временную базу для всех действий на этом узле. Логические часы базируются на аппаратных часах своего узла, хотя обычно они имеют гораздо большую степень детализации, чем аппаратные часы. Алгоритм, выполняемый каждым узлом для синхронизации логических часов, можно рассматривать как процесс синхронизации, вызываемый в конце каждого интервала повторной синхронизации, который отвечает за периодическое считывание значений часов в других узлах, а затем за настройку соответствующего значения локальных часов [18].

Принцип аппаратных алгоритмов синхронизации основан на принципе фазовой автоподстройки частоты. Аппаратные часы в каждом узле — это выход генератора, управляемого напряжением. Напряжение, подаваемое на генератор, поступает от фазового детектора, выходное значение которого пропорционально фазовой ошибке между фазой его тактового сигнала (выход управляемого напряжением генератора, которым он управляет) и опорным сигналом, создаваемым с помощью других тактовых сигналов в системе. Таким обра-

зом, часы всегда могут быть синхронизированы друг с другом посредством регулирования частоты каждого отдельного тактового сигнала в соответствии с эталонным сигналом [18].

Под аппаратными часами [19] понимаются осцилляторы, которые «тикают» с определенной частотой. Эти «тики» можно пересчитать. Поскольку изменение частоты «тиканья» аппаратных часов обходится дорого, используется понятие виртуальных часов. Виртуальные часы работают следующим образом: целое число «тиков» аппаратных часов составляет один «тик» виртуальных часов. Для коррекции (ускорения или замедления) виртуальных часов изменяется количество «тиков» аппаратных часов, приходящихся на «тик» виртуальных часов. Такая адаптация используется для синхронизации виртуальных часов различных узлов.

Для того чтобы устранить внутренние ограничения как аппаратного, так и программного подходов, авторы [20] предложили гибридную схему синхронизации, которая обеспечивает баланс между достижимыми перекосами часов и требованиями к аппаратному обеспечению. Их схема особенно хорошо подходит для больших, частично связанных однородных распределенных систем с топологиями межточечных связей, таких как гиперкубы или сетки.

В заданное время в интервале повторной синхронизации временной процесс передает локальное значение времени всем другим временным процессам. Широковещательный алгоритм работает таким образом, что все временные процессы получают несколько копий сообщения о времени по непересекающимся путям. Количество копий, используемых в широковещательном алгоритме, зависит от максимального числа допустимых неисправностей и модели неисправности системы. Когда временной процесс получает сообщение, отправленное каким-либо другим временным процессом, он записывает время (в соответствии со своими локальными часами), в которое было получено сообщение. Затем в соответствии с широковещательным алгоритмом он передает сообщение другим временным процессам.

Перед ретрансляцией сообщения временной процесс добавляет к сообщению время, прошедшее (согласно его собственным часам) с момента получения сообщения. В конце интервала повторной синхронизации он вычисляет расхождения между локальными часами и часами исходного узла для каждой полученной копии. Затем процесс синхронизации выбирает $(m + 1)$ наибольшее значение в качестве оценки расхождения между двумя часами. В качестве поправки к локальным часам используется среднее значение расчетных расхождений по всем узлам. Требуется минимум $(3m + 1)$ узлов, чтобы выдержать m византийских неисправностей [18].

Работы [12, 16, 17] продолжены авторами [21], которые разработали новый отказоустойчивый алгоритм решения одного из вариан-

тов задачи синхронизации часов Л. Лэмпорта. Алгоритм предназначен для системы распределенных процессов, которые взаимодействуют посредством отправки сообщений. Каждый процесс имеет свои собственные физические часы, доступные только для чтения, скорость дрейфа которых от реального времени очень мала. Добавляя их значение к своему физическому часовому времени, процесс получает текущее местное время. С помощью алгоритма, устойчивого к неисправности чуть менее трети участвующих процессов, решается задача поддержания точно синхронизированного локального времени, в предположении, что локальное время процессов изначально точно синхронизировано. Он поддерживает синхронизацию с точностью до небольшой постоянной, величина которой зависит от скорости дрейфа часов, времени доставки сообщения и его неопределенности, а также от точности начальной синхронизации. В [21] дается характеристика того, как далеко часы дрейфуют от реального времени.

Алгоритм синхронизации часов, предложенный в [21], должен справляться с техническими сбоями и восстановлением, дрейфом часов и изменением времени доставки сообщений, но эти условия усложняют разработку и анализ алгоритмов. Авторы [21] разработали алгоритм, который удовлетворяет перечисленным требованиям, предполагая, что часы изначально достаточно хорошо синхронизированы и что неисправны менее трети процессов, причем в модели учитывается, что процессы имеют доступ только для чтения к локальным физическим часам, которые подвержены очень малой скорости дрейфа. Коммуникационная сеть полносвязная, поэтому каждый процесс может передавать сообщение непосредственно всем другим процессам одновременно. Все сообщения доставляются в течение фиксированного промежутка времени с небольшой погрешностью. Модель не требует наличия неподделываемых подписей.

Алгоритм из [21], как и алгоритм в [12], работает в раундах. Значения времени накапливаются в каждом раунде и усредняются с помощью отказоустойчивой функции усреднения, аналогичной приведенной в [22] для расчета корректировки. Функция отбрасывает самое большое и самое маленькое значения $f_{\min}$ и $f_{\max}$, затем применяет некоторую обычную функцию усреднения к оставшимся значениям.

Особенность практического алгоритма синхронизации часов состоит в возможности восстановления часов сбившегося процесса путем синхронизации его часов с другими процессами.

Проблема состоит в поддержке синхронизации местного времени, как только она будет установлена. Кроме того, существует проблема установления начальной синхронизации. Отказоустойчивая функция усреднения, используемая в алгоритме обслуживания [21], вдвое уменьшает ошибку в каждом раунде, поэтому ее можно ис-

пользовать для синхронизации часов, которые начинаются с произвольных значений. Предложенный в [21] вариант алгоритма можно использовать для начальной синхронизации в условиях дрейфа часов, неопределенности времени доставки сообщений и произвольных сбоев процесса.

В алгоритме [21] имеются следующие ограничения на значения параметров. В реальной системе параметры ρ (скорость дрейфа), δ (средняя задержка сообщения) и έ (неопределенность задержки) фиксируются используемыми аппаратными средствами и протоколами низкоуровневой связи. Разработчик алгоритмов имеет некоторую свободу при выборе R (длина раунда) и β (рассогласование, т. е. насколько точно в реальном времени процессы выполняются в рамках одного и того же раунда) при допустимости предположения, что часы изначально начинают алгоритм в пределах β. Для того чтобы часы были максимально точно синхронизированы, β должно быть как можно меньше. Однако чем меньше β, тем меньше должно быть R, т. е. тем чаще мы должны синхронизировать часы.

Однако R не может быть сколь угодно малым, этот параметр должен быть достаточно большим, чтобы алгоритм работал правильно, для этого необходимо обеспечить следующее:

1) после того как исправный процесс w сбрасывает свои часы, местное время, в которое w планирует следующую трансляцию, должно быть больше, чем местное время на часах в момент сброса;

2) сообщение, отправленное исправным процессом w в раунде r (r — номер раунда), которое будет использоваться для установки $(r + 1)$-х логических часов, поступает в исправный процесс q после того, как q уже установил свои r-е логические часы.

Все это в совокупности может послужить препятствием при достижении синхронизации в сбое- и отказоустойчивых многокомплексных РМВС ответственного применения, несмотря на то что авторы [21] декларируют (хотя это и не является очевидным на данном этапе анализа), что достаточные условия, связывающие параметры, имеют вид

$$R > 2(1+\rho)(\beta+\acute{\varepsilon}) + (1+\rho)\max\{\delta,\ \beta+\acute{\varepsilon}\}) + \rho\delta;$$

$$R \le \frac{\beta}{4\rho} - \frac{\acute{\varepsilon}}{\rho} - \rho(\beta+\delta+\acute{\varepsilon}) - 2\beta - \delta - 2\acute{\varepsilon},$$

из чего следует, что

$$\beta \ge 4\acute{\varepsilon} + 4\rho\left(4\beta+\delta+4\acute{\varepsilon}+\max\{\delta,\ \beta+\acute{\varepsilon}\}\right) +$$

$$+4\rho^2\left(3\beta+2\delta+3\acute{\varepsilon}+\max\{\sigma,\ \beta+\acute{\varepsilon}\}\right).$$

По утверждению авторов [21], любая комбинация R и β, которая удовлетворяет этим неравенствам, будет работать в предложенном ими алгоритме. Если R считать фиксированным, то точность синхронизации по оси реального времени β составляет примерно $4\varepsilon + 4\rho R$. Такое значение получается решением неравенств для R относительно β, пренебрегая элементами порядка ρ. Кроме того, отказоустойчивая функция усреднения не гарантирует оптимальных значений максимальной коррекции, максимального дрейфа и максимального отклонения параметров синхронизации.

В [23] рассмотрена жесткая внутренняя синхронизация часов, которая требует выполнения двух условий:

1) в любое время рассогласование между двумя правильными часами должно быть ограничено постоянной величиной β (максимальное отклонение или рассогласование);

2) скорость дрейфа часов относительно реального времени τ должна быть ограничена постоянной величиной τ_v.

Синхронизация часов является нетривиальной проблемой из-за необходимости учитывать сбои, поскольку возникновение сбоя в момент синхронизации часов способно нарушить процесс синхронизации.

Большинство алгоритмов синхронизации можно описать как экземпляры одного абстрактного, общего алгоритма синхронизации часов, используя понятие функции сходимости, введенное в [24]. Этот общий алгоритм заключается в следующем: в конце каждого цикла синхронизации каждый процесс считывает часы всех процессов, затем корректирует свое значение часов для следующего цикла, применяя функцию сходимости к показаниям часов текущего цикла. Алгоритмы синхронизации, которые могут быть получены из этого общего алгоритма путем создания экземпляра некоторой конкретной функции для абстрактного понятия функции сходимости, названы алгоритмами, основанными на функции сходимости [23].

В любой момент времени значение синхронизированных часов определяется как сумма текущего значения аппаратных часов и корректирующей переменной. Поскольку аппаратные часы дрейфуют отдельно от реального времени и друг от друга, переменная настройки часов должна периодически обновляться. Максимальное изменение значения времени, которое алгоритм синхронизации может вызвать во время настройки, называется максимальной коррекцией. Настройки часов вместе с дрейфом аппаратных часов приводят к ошибке, когда синхронизированные часы используются для измерения временного интервала. Эта ошибка может быть ограничена постоянной частью, которая называется максимальным разрывом, и частью, которая зависит от длины измеряемого интервала времени. Эта вторая часть называется максимальной скоростью дрейфа синхронизированных часов [23].

Когда часы настраиваются дискретно, т. е. переменные настройки меняются только один раз за раунд, синхронизированные часы не могут быть гарантированно монотонными. При использовании функции сходимости с оптимальной максимальной поправкой немонотонность точно синхронизированных часов становится ничтожно малой, так как время считывания часов обычно больше, чем оптимальная максимальная коррекция. Предложенная в [23] функция сходимости полезна для алгоритмов синхронизации, основанных на статистическом дистанционном считывании часов, описанном в [25], и обеспечивает, в отличие от [21], оптимальность максимальной коррекции, максимальной скорости дрейфа и максимального отклонения.

Поскольку статистические методы считывания не обеспечивают априорной или вычисленной верхней границы ошибки, допущенной при считывании удаленных часов, синхронизированные часы могут многократно корректироваться вперед и назад с большими поправками из-за чрезмерных ошибок считывания часов, это может служить препятствием к достижению взаимного информационного согласования в распределенных системах.

Кроме того, в [23] рассмотрены только алгоритмы внутренней синхронизации часов, способные маскировать произвольные сбои часов и технические сбои. Когда авторы говорят об алгоритме синхронизации, то они раскрывают процесс внутренней синхронизации часов, устойчивый к произвольным сбоям, но не упоминают об устойчивости этих алгоритмов к отказам.

Синхронизация часов может быть достигнута либо аппаратным, либо программным обеспечением. Аппаратная временная синхронизация [26] обеспечивает очень точную синхронизацию за счет использования специального оборудования синхронизации на каждом процессоре и использует отдельную сеть исключительно для временных сигналов. Программные алгоритмы синхронизации часов [6, 12, 21, 27–36] используют стандартные коммуникационные сети и отправляют сообщения синхронизации для синхронизации часов. Они не требуют специального оборудования, но и не обеспечивают настолько точную синхронизацию, как это делают аппаратные алгоритмы. Программные алгоритмы синхронизации часов подразделяются на следующие типы:

– детерминированные;
– вероятностные;
– статистические.

В детерминированных алгоритмах предполагается, что существует верхняя граница задержки передачи. Они гарантируют и наличие верхней границы разности между любыми двумя значениями времени (точность). Используя вероятностные и статистические алгоритмы

синхронизации часов, нельзя сделать предположения о максимальных задержках сообщений. В вероятностных алгоритмах нет информации о распределениях задержки, в статистических алгоритмах математическое ожидание и стандартное отклонение распределений задержки известны. В отличие от детерминированных алгоритмов синхронизации часов, вероятностные и статистические алгоритмы гарантируют постоянное максимальное отклонение между любыми значениями часов с вероятностью строго меньше единицы. В вероятностных алгоритмах часы всегда «знают», синхронизированы они с другими или нет, тогда как в статистических алгоритмах часам «неизвестно», насколько точно их значения соответствуют друг другу.

Синхронизация часов широко изучалась в течение многих лет. Подробные исследования можно найти в [18, 37, 38]. В [18] программные и аппаратные алгоритмы синхронизации часов классифицируются в зависимости от используемой схемы коррекции часов. Работа [37] дает единую объединяющую парадигму и доказательство правильности, которые могут быть использованы для понимания в основном всех детерминированных алгоритмов синхронизации часов, поддерживающих византийские неисправности [5]. Алгоритмы, рассмотренные в [38], представлены в соответствии с поддерживаемыми ошибками и системной синхронностью, т. е. знанием верхних границ задержек связей.

В работе [39] обсуждается ряд распределенных алгоритмов, которые используют синхронизированные часы, и анализируется, как эти часы используются в предлагаемых алгоритмах. По мнению автора [39], синхронизированные часы интересны тем, что их можно применять для улучшения производительности распределенной системы, поскольку они позволяют заменить обмен локальными вычислениями. Например, вместо запроса со стороны узла N к другому узлу M о выполнении какого-либо свойства, он может вывести ответ на основе некоторой информации о M из прошлого вместе с текущим временем на часах N, а также определяется роль синхронизированных алгоритмов.

По алгоритмам синхронизации часов имеется обширная литература, о чем свидетельствует наличие обзоров [5, 38], поэтому в [39] предполагается, что синхронизация часов существует, подробно описана и известно, как ее использовать.

Алгоритмы синхронизации часов основаны на вероятностных предположениях о тактовой частоте и задержке сообщения. Следовательно, часы синхронизируются только с некоторой, хотя и очень высокой вероятностью. Поскольку синхронизация часов может иногда давать сбой, наиболее желательно, чтобы от синхронизации зависела производительность, но не правильность алгоритмов. Часть обсужда-

емых в [39] алгоритмов действительно зависит от синхронизации только в части производительности, но есть и такие, что зависят от нее в плане правильности. В практических системах производительность очень важна. Кроме того, правильность алгоритма может зависеть от возникновения других маловероятных событий, так что их зависимость от синхронизации часов не окажет большого влияния. Существуют также механизмы восстановления на более высоком уровне для компенсации сбоев алгоритма. Однако в сбое- и отказоустойчивых РМВС ответственного применения необходимо гарантировать правильность работы алгоритмов в любых ситуациях.

Заключение. Для разработки современных сбое- и отказоустойчивых РМВС ответственного применения требуются действия, связанные с синхронизацией часов. Актуальность вопросов синхронизации многократно возрастает одновременно с увеличением сложности архитектуры РМВС.

В первой части работы приведены основные определения, связанные с процессом синхронизации, рассмотрены программные и аппаратные методы синхронизации, их особенности, достоинства и недостатки. Анализ литературных источников показал, что для устранения внутренних ограничений как аппаратного, так и программного подхода целесообразно использовать гибридную схему синхронизации, которая обеспечивает баланс между достижимыми характеристиками часов и требованиями к аппаратному обеспечению.

Рассмотрены также вопросы, связанные с дрейфом часов, и сложности, связанные с необходимостью учитывать сбои, поскольку возникновение сбоя в момент синхронизации часов способно нарушить процесс синхронизации, что может привести к нарушению работы системы в целом.

ЛИТЕРАТУРА

[1] Dolev D., Dwork C., Stockmeyer L. On the minimal synchronics needed for distributed consensus. *Proceedings 24th Annual Symposium on Foundations of Computer Science. November 7–9, 1983, IEEE, Tucson, USA*. IEEE, 1983, pp. 393–402.

[2] Лобанов А.В., Сиренко В.Г. Проблема отказоустойчивости в сетецентрических информационно-управляющих системах. *Образовательные ресурсы и технологии*, 2014, № 2 (5), с. 115–121.

[3] Эйкхофф Й. *Бортовые компьютеры, программное обеспечение и полетные операции. Введение*. Москва, Техносфера, 2018, 334 с.

[4] Lamport L. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 1978, vol. 21, no. 7, pp. 558–565.

[5] Anceaume Em., Puaut I. *A taxonomy of clock synchronization algorithms*. Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA), Research Report 1103, 1997. URL: http://citeseer.ist.psu.edu/anceaume97taxonomy.html

[6] Cristian F., Aghili H., Strong R., Dolev D. Atomic broadcast: from simple message diffusion to byzantine agreement. *Proceedings of 15th International Sym-*

posium on Fault-Tolerant Computing Systems. *June 29*, *1985*, *IEEE*, *Ann Arbor-USA*. IEEE, 1985, pp. 200–206.

[7] Lamport L., Shostak R., Pease M. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems,* 1982, vol. 4, no. 3, pp. 382–401.

[8] Ковязина Д.Р. Автоматическая калибровка и синхронизация времени во встраиваемых вычислительных системах. *Научно-технический вестник Санкт-Петербургского государственного университета информационных технологий, механики и оптики*, 2009, № 2 (60), с. 87–93.

[9] *Словарь русских синонимов*. URL: https://dic.academic.ru/dic.nsf/dic_fwords/10911 (дата обращения 03.02.2021).

[10] IEEE Std 1588™–2008. Revision of IEEE Std 1588–2002. *IEEE Instrumentation and Measurement Society*, 2008, 269 p.

[11] Eidson J.C. Measurement, control and communication using. *IEEE* 1588 (Advances in Industrial Control), Springer, 2006, 283 p.

[12] Lamport L., Melliar-Smith P.M. Synchronizing clocks in the presence of faults. *Journal of the ACM*, 1985, vol. 32, no. 1, pp. 52–78.

[13] Lamport L. The implementation of reliable distributed multiprocess systems. *Computer Network*, 1978, vol. 2, pp. 95–114.

[14] Lamport L. Using time instead of timeout for fault-tolerant distributed systems. *ACM Transactions on Programming Languages and Systems*, 1984, vol. 6, no. 2, pp. 254–280. DOI:10.1145/2993.2994

[15] Dolev D., Halpern J.Y., Strong R. On the possibility and impossibility of achieving clock synchronization. *Proceedings of the 16th Annual ACM STOC. April 30 — May 2, 1984, Washington D.C., USA.* ACM, New York, 1984, pp. 504–511.

[16] Halpern J., Simons B., Strong R. An efficient fault-tolerant algorithm for clock synchronization. *Proceedings of the 3rd annual ACM symposium on Principles of distributed computing*, *August 27–29, 1984, Vancouver, B. C., Canada*. Association for Computing Machinery, New York, 1984, pp. 75–88. DOI: 10.1145/800222.806738

[17] Dolev D., Halpern J.Y., Strong H.R. On the possibility and impossibility of achieving clock synchronizati on. *Proceedings of 16th Annual ACM Symposium on Theory of Computing*, *April, 30 — May, 2, 1984, Washington, D.C*. ACM, New York, 1984, pp. 504–511.

[18] Ramanathan P., Shin K.G., Butler R.W. Fault-tolerant clock synchronization in distributed systems. *Computer*, 1990, vol. 23, no. 10, pp. 33–42. DOI: 10.1109/2.58235

[19] Paulitsch M., Steiner W. Fault-tolerant clock synchronization for embedded distributed multi-cluster systems. *Proceedings of 15th Euromicro Conference "Real-Time Systems"*, *July, 2–4, 2003, Porto, Portugal.* New York, IEEE, 2003, pp. 249–256. DOI: 10.1109/EMRTS.2003.1212750

[20] Ramanathan P., Kandlur D.D., Shin K.G. Hardware-assisted software clock synchronization for homogeneous distributed systems*, IEEE Transactions on Computers,* 1990, vol. C-39, no. 4, pp. 514–524.

[21] Welch L.J., Lynch N. A new fault-tolerant algorithm for clock synchronization. *Information and computation*, 1988, vol. 77, iss. 1, pp. 1–36. DOI: 10.1016/0890-5401(88)90043-0

[22] Dolev D., Lynch N., Pinter S., Stark E., Weihl W. Reaching approximate agreement in the presence of faults. *Proceedings of the 3rd Annual IEEE Symposium on Distributed Software and Database Systems, October, 26–29, 1983, Association for Computing Machinery, New York.* New York, IEEE, 1983, pp. 145–154.

[23] Fetzer C., Cristian F. An optimal internal clock synchronization algorithm. *Proceedings of the 10th Annual Conference "Computer Assurance. COMPASS — 95*, *Systems Integrity, Software Safety and Process Security*, *June 26–30*, *1995*, *IEEE*. New York, IEEE, 1995, pp. 187–196. DOI: 10.1109/CMPASS.1995.521898

[24] Schneider F. Understanding protocols for Byzantine clock synchronization. *Technical Report*, 1987, p. 859.

[25] Mills D.L. Internet time synchronization: the network time protocol. *IEEE Transactions on Computers*, 1991, vol. 39, no. 10, pp. 1482–1493.

[26] Kopetz H., Ochsenreiter W. Clock synchronization in distributed real-time computer systems. *IEEE Transactions on Computers,* 1987, C-36, no. 8, pp. 933–940.

[27] Halpern J., Strong H., Simons B., Dolev D. Fault-tolerant clock synchronization. *Proceedings of 3rd International Symposium on Principles of Distributed Computing*, *August 27–29*, *1984*, *Vancouver*, *B.C.*, *Canada*. Association for Computing Machinery, New York, 1984, pp. 89–102. DOI: 10.1145/800222.806739

[28] Srikanth T.K., Toueg S. Optimal clock synchronization. *Journal of the ACM,* 1987, vol. 34, no. 3, pp. 626–645.

[29] Verissimo P., Casimiro A., and Rodrigues L. Cesiumspray: a precise and accurate global time service for large scale systems. *Journal of Real-Time Systems,* 1997, vol. 12, pp. 243–294.

[30] Pfluegl M., Blough D. A new and improved algorithm for fault-tolerant clock synchronization. *Journal of Parallel and Distributed Computing*, 1995, vol. 27, pp. 1–14.

[31] Gusella R., Zatti S. The accuracy of the clock synchronization achieved by TEMPO in berkeley UNIX 4.3BSD. *IEEE Transactions on Software Engineering*, 1989, vol. 15, no. 7, pp. 847–853.

[32] Mahaney S., Schneider F. Inexact agreement: Accuracy, precision and graceful degradation. *Proceedings of 4th International Symposium on Principles of Distributed Computing, August 5–7*, *1985*, *Minaki*, *Ontario*, *Canada*. New York, Association for Computing Machinery, 1985, pp. 237–249. DOI: 10.1145/323596.323618

[33] Fetzer C., Cristian F. Lower bounds for function based clock synchronization. *Proceedings of 13th International Symposium on Principles of Distributed Computing*, *August 18–21*, *1995*, *Ottowa*, *Ontario*, *Canada*. New York, Association for Computing Machinery, 1995, pp. 137–143. DOI: 10.1145/224964.224980

[34] Fetzer C., Cristian F. Integrating external and internal clock synchronization. *Journal of Real-Time Systems*, 1997, vol. 12, no. 2, pp. 123–172.

[35] Arvind K. Probabilistic clock synchronization in distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 1994, vol. 5, no. 5, pp. 474–487.

[36] Cristian F. Probabilistic clock synchronization. *Distributed Computing*, 1989, vol. 3, pp. 146–158.

[37] Schneider F. A paradigm for reliable clock synchronization. *Technical Report TR86-735*. Computer Science Department. Ithaca, NY, Cornell University, 1986. URL: https://hdl.handle.net/1813/6575 (дата обращения 03.02.2021).

[38] Simons B., Lundelius-Welch J., Lynch N. An overview of clock synchronization. *Asilomar Workshop on Fault-tolerant Distributed Computing Conference*, 1990, vol. 448, pp. 84–96.

[39] Liskov B. Practical uses of synchronized clocks in distributed systems. *Distributed Computing*, *Periodicals Distributed Computing*, 1993, vol. 6, no. 4, pp. 211–219. https://doi.org/10.1007/BF02242709

Статья поступила в редакцию 17.03.2021

Ссылку на эту статью просим оформлять следующим образом:

Ашарина И.В. Синхронизация часов в распределенных многомашинных вычислительных системах. Часть I. *Инженерный журнал: наука и инновации*, 2021, вып. 4. http://dx.doi.org/10.18698/2308-6033-2021-4-2074

Ашарина Ирина Владимировна — канд. техн. наук, доцент, старший научный сотрудник АО «НИИ «Субмикрон». e-mail: asharinairina@mail.ru

Clock synchronization in distributed multicomputer systems. Part I

© I.V. Asharina

JSC Research Institute Submicron, Moscow, Zelenograd, 124460, Russia

The study substantiates the necessity of clock synchronization in distributed multicomputer systems. The basic definitions related to the concept of clock synchronization are given, and methods of clock synchronization are classified. Increasing the lifecycle of failure- and fault-tolerant distributed multicomputer systems for critical application is one of the most urgent problems at the current level of technology development. This is especially true for unattended distributed multicomputer systems for space applications. The development of such systems should begin with the construction of models of faults and self-controlled degradation, ensuring, firstly, their failure and fault tolerance and, secondly, maximum survivability, which is possible only if there are means of clock synchronization in such systems. All activities associated with ensuring the synchronization of any distributed multicomputer systems begin with the concept of synchronization of on-board functions, which is based on the generation of on-board time and includes the synchronization of on-board software and equipment that requires time synchronization or information about the course of time. The main elements of this concept are the processor clock module, the onboard software clock, the atomic navigation clock.
The first part of the work gives basic definitions, and considers methods and algorithms related to the clock synchronization process. The second part is devoted to synchronization in systems with Byzantine faults and in multi-cluster and multi-complex, i.e. multitask, systems. The modern technologies providing the synchronization process in such systems are considered.

Keywords: *distributed multicomputer system, failure and fault tolerance, clock synchronization, dynamic redundancy, Byzantine fault*

REFERENCES

[1] Dolev D., Dwork C., Stockmeyer L. On the minimal synchronics needed for distributed consensus. *Proc. 24th Symp. on Foundationcs of Computer Science.* IEEE, Tucson, USA, 1983, pp. 393–402.

[2] Lobanov A.V., Sirenko V.G. *Obrazovatelnye resursy i tekhnologii* (*Educational resources and technologies*), 2014, no. 2 (5), pp. 115–121.

[3] Eickhoff J. *On-board computers, onboard software and satellite operations: An Introduction*. Springer, 2016, 300 p. [In Russ.: Eickhoff J. Bortovye komp'iutery, programmnoe obespechenie i poletnye operatsii. Vvedenie. Moscow, Tekhnosfera Publ., 2018, 344 p.].

[4] Lamport L. Time, clocks, and the ordering of events in a distributed system, Comm. *ACM*, July 1978, vol. 21 (7), pp. 558–565.

[5] Anceaume Em., Puaut I. *A taxonomy of clock synchronization algorithms*. Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA), Research Report 1103, 1997. Available at: http://citeseer.ist.psu.edu/anceaume97taxonomy.html

[6] Cristian F., Aghili H., Strong R., Dolev D. Atomic broadcsat: from simple message diffusion to byzantine agreement. *Proc. 15th Int. Symp. on Fault-Tolerant Computing Systems*, 1985.

[7] Lamport L., Shostak R., Pease M. The byzantine generals problem. *ACM Trans. Progr. Lang. Syst.,* 1982, vol. 4, no. 3, pp. 382–401.

[8] Kovyazina D.R. *Nauchno-tekhnicheskiy vestnik Sankt-Peterburgskogo gosudarstvennogo universiteta informatsionnykh tekhnologii, mekhaniki i optiki — Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2009, no. 2 (60), pp. 87–93.

[9] *Slovar russkikh sinonimov* [Dictionary of Russian synonyms]. Available at: https://dic.academic.ru/dic.nsf/dic_fwords/10911

[10] IEEE Std 1588™-2008 (Revision of IEEE Std 1588-2002). *IEEE Instrumen. Measur. Soc.,* 2008, 269 p.

[11] Eidson J.C. Measurement, Control and Communication Using. *IEEE 1588*, Springer, 2006, 283 p.

[12] Lamport L., Melliar-Smith P.M. Synchronizing clocks in the presence of faults. *J. Assoc. Comput. Mach*., January 1985, no. 32 (1), pp. 52–78.

[13] Lamport L. The implementation of reliable distributed multiprocess systems. *Comput. Netw.,* 1978, no. 2, pp. 95–114.

[14] Lamport L. Using time instead of timeout for fault-tolerant distributed systems. *ACM Trans. Prog. Lang. Syst,* April 1984. https://doi.org/10.1145/2993.2994

[15] Dolev D., Halpern J.Y., Strong R. On the possibility and impossibility of achieving clock synchronization. *Proc. 16th Annual ACM STOC* (Washington D.C., Apr.). ACM, New York, 1984, pp. 504–511. (Also to appear in *J. Comput. Syst. Sc.*).

[16] Halpern J., Simons B., Strong R. An efficient fault-tolerant algorithm for clock synchronization. *Proc. 3rd Annual ACM Symp. on Principles of Distributed Computing*, *August 27–29*, *1984*, *Vancouver*, *B.C.*, *Canada.* Association for Computing Machinery, New York, 1984, pp. 75–88. DOI: 10.1145/800222.806738

[17] Dolev D., Halpern J.Y., Strong H.R. On the possibility and impossibility of achieving clock synchronization. *Proc. 16th Annual ACM Symp. on Theory of Computing* (Washington, D.C., Apr. 30 — May 2). ACM, New York, 1984, pp. 504–511.

[18] Ramanathan P., Shin K.G., Butler R.W. Fault-Tolerant Clock Synchronization in Distributed Systems. *Computer*, November 1990, no. 23 (10), pp. 33–42. DOI: 10.1109/2.58235

[19] Paulitsch M., Steiner W. Fault-Tolerant Clock Synchronization for Embedded Distributed Multi-Cluster Systems. *Conference: Real-Time Systems*, *August 2003. Proc. 15th Euromicro Conf.* Technische Universität Wien, Vienna, Austria. DOI: 10.1109/EMRTS.2003.1212750

[20] Ramanathan P., Kandlur D.D., Shin K.G. Hardware-Assisted Software Clock Synchronization for Homogeneous Distributed Systems. *IEEETrans. Computers*, Apr. 1990, vol. C-39, no. 4, pp. 514–524.

[21] Welch L.J., Lynch N. A new fault-tolerant algorithm for clock synchronization. *Information and Computation*, 1988, no. 77, pp. 1–36. DOI: 10.1016/0890-5401(88)90043-0

[22] Dolev D., Lynch N., Pinter S., Stark E., Weihl W. Reaching approximate agreement in the presence of faults. *Proc. 3rd Annual IEEE Symp. on Distributed Software and Database Systems*, *October 26–29*, *1983*, *Association for Computing Machinery, New York*. New York, IEEE, 1983, pp. 145–154.

[23] Fetzer Ch., Cristian F. An Optimal Internal Clock Synchronization Algorithm. *Conference: Computer Assurance. COMPASS '95. Systems Integrity, Software Safety and Process Security. Proceedings of the Tenth Annual Conference*, 1995, pp. 187–196. DOI: 10.1109/CMPASS.1995.521898

[24] Schneider F. Understanding protocols for Byzantine clock synchronization. *Technical Report*, Dept of Computer Science, Cornel1 University, 1987, 859 p.

[25] Mills D.L. Internet time synchronization: the network time protocol. *IEEE Truns. Communications*, October 1991, no. 39 (10), pp. 1482–1493.

[26] Kopetz H., Ochsenreiter W. Clock synchronization in distributed real-time computer systems. *IEEE Transactions on Computers,* August 1987, C-36(8), pp. 933–940.
[27] Halpern J., Strong H., Simons B., Dolev D. Fault-tolerant clock synchronization. *Proceedings of 3rd International Symposium on Principles of Distributed Computing, August 27–29, 1984, Vancouver, B.C., Canada*. Association for Computing Machinery, New York, 1984, pp. 89–102. DOI: 10.1145/800222.
[28] Srikanth T.K., Toueg S. Optimal clock synchronization. *Journal of the ACM*, July 1987, no. 34 (3), pp. 626–645.
[29] Verissimo P., Casimiro A., Rodrigues L. Cesiumspray: a precise and accurate global time service for large scale systems. *Journal of Real-Time Systems,* 1997, no. 12, pp. 243–294.
[30] Pfluegl M., Blough D. A new and improved algorithm for fault-tolerant clock synchronization. *Journal of Parallel and Distributed Computing,* 1995, no. 27, pp. 1–14.
[31] Gusella R., Zatti S. The accuracy of the clock synchronization achieved by TEMPO in Berkeley UNIX 4.3BSD. *IEEE Transactions on Software Engineering,* July 1989, no. 15 (7), pp. 847–853.
[32] Mahaney S., Schneider F. Inexact agreement: Accuracy, precision and graceful degradation. *Proc. 4th Int. Symp. on Principles of Distributed Computing*, *August 5–7*, *1985*, *Minaki*, *Ontario*, *Canada.* New York, Association for Computing Machinery, 1985, pp. 237–249. DOI: 10.1145/323596.323618
[33] Fetzer C., Cristian F. Lower bounds for function based clock synchronization. *Proc. 13th Int. Symp. on Principles of Distributed Computing*, *August 18–21*, *1995*, *Ottowa*, *Ontario*, *Canada.* New York, Association for Computing Machinery, 1995, pp. 137–143. DOI: 10.1145/224964.224980
[34] Fetzer C., Cristian F. Integrating external and internal clock synchronization. *Journal of Real-Time Systems*, 1997, no. 12 (2), pp. 123–172.
[35] Arvind K. Probabilistic clock synchronization in distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 1994, no. 5 (5), pp. 474–487.
[36] Cristian F. Probabilistic clock synchronization. *Distributed Computing*, 1989, vol. 3, pp. 146–158.
[37] Schneider F. A paradigm for reliable clock synchronization. *Technical Report TR86-735*. Computer Science Department. Cornell University, February 1986. Available at: https://hdl.handle.net/1813/6575 (accessed February 3, 2021).
[38] Simons B., Lundelius-Welch J., Lynch N. An overview of clock synchronization. In: Simons B., Spector A., eds. *Fault-Tolerant Distributed Computing. Lecture Notes in Computer Science*, 1990, vol. 448, pp. 84–96.
[39] Liskov B. Practical Uses of Synchronized Clocks in Distributed Systems. *Distributed Computing. Periodicals Distributed Computing*, vol. 6, no. 4, pp. 211–219. https://doi.org/10.1007/BF02242709

Asharina I.V., Cand. Sc. (Eng.), Assoc. Professor, Senior Research Fellow, JSC Research Institute Submicron. e-mail: asharinairina@mail.ru