# Оптимизация доступа к страницам памяти в системах, использующих программную реализацию глобального страничного кеша

*Е.И. Гусев <eugene@levelextra.ru>*
*Национальный технический университет Украины*
*«Киевский политехнический институт имени Игоря Сикорского»,*
*Украина, 03056, г. Киев - 56, проспект Победы, 37*

**Аннотация**. В статье рассматривается способ обработки распределённых страниц в Oracle Real Application Clusters (Oracle RAC) и проводится его сравнение с другими известными способами в контексте сравнения архитектур доступа к страницам. В результате выявления недостатков традиционного способа, применяемого в Oracle RAC, предлагается новый способ доступа, в основе которого лежит введение еще одного состояния страницы – состояния «разгрузки», повышающее эффективность обработки распределённых страниц за счёт снижения количества пересылок между узлами при обработке горячих страниц.

**Ключевые слова:** Oracle RAC; распределённая страница; shared nothing; shared everthing; Global Cache Fusion; СУБД

**DOI:** 10.15514/ISPRAS-2018-30(1)-11

**Для цитирования:** Гусев Е.И. Оптимизация доступа к страницам памяти в системах, использующих программную реализацию глобального страничного кеша. Труды ИСП РАН, том 30, вып. 1, 2018 г., стр. 161-182. DOI: 10.15514/ISPRAS-2018-30(1)-11

## *1. Введение*

Данная работа посвящена изложению одной из основных идей диссертации [1], защищённой в июне 2017 года. Распространение технологий облачных вычислений вывело проблему общего ресурса в контексте масштабирования систем управления базами данных (СУБД) на новый уровень. Облачные технологии для большинства вычислительных задач (включая рендеринг, трансформацию и т. д.) привнесли и снижение стоимости ресурсов и упрощение администрирования за счёт консолидации. Однако необходимо отметить, что для СУБД ими решается только одна задача – повышение эффективности управления, в ущерб стоимости ресурсов, так как требования к





мощности при консолидации возрастают, а возможности использовать для консолидации дешёвые облачные технологии отсутствует. Главная причина здесь – неэффективные технологии обработки общего ресурса, не учитывающие особенности облачных систем и, как следствие, не масштабирующиеся.

Под масштабированием СУБД мы в данном случае будем понимать трансформацию одноузловой системы в многоузловую. Невозможность пропорционального увеличения мощности базы данных (БД) добавлением в облако серверов решается, как правило, консолидацией всех общих ресурсов в единую центральную СУБД и наращиванием её аппаратной мощности. Результатом сложившейся практики является ситуация, когда БД в системах облачных вычислений становится общим ресурсом, используемым всеми вычислительными мощностями.

Другой, довольно часто применяемый приём, – существенное изменение архитектуры приложений для снижения интенсивности обращений к БД часто за счёт снижения контроля за целостностью данных. Яркая иллюстрация такого случая – NoSQL DB [2-7].

Применение же классических способов блокирования при масштабировании облачных (впрочем, и других распределённых) СУБД приводит к значительному увеличению времени обработки и блокирования по сравнению с одноузловыми системами.

Всё это обуславливает кризис в базах данных, выражающийся как в появлении новых архитектур, тесно связанных с приложениями (как, например, NoSQL DB [2-7]) для разгрузки БД от запросов либо облегчения самого запроса, так и новом витке развития дорогостоящих программно-аппаратных комплексов (Oracle Exadata [8], IBM DB2 pureScale [9]). Собственно, невысокие возможности облачных систем консолидировать системы с общим ресурсом и обусловили кризис в развитии СУБД в контексте развития систем облачных вычислений.

Рассматривая модели обслуживания облачных систем, следует отметить, что предоставление СУБД как сервиса клиенту подходит под классификацию «платформа как услуга», или PaaS (platform as a service) [10-12]. Хотя некоторые исследователи относят облачные СУБД к SaaS (software as a service) [13], это, учитывая определение, данное в [11], неверно, поскольку в определении чётко говорится о «несложных покупке и сопровождении программного обеспечения и инфраструктуры, лежащей в основе сервиса».

Среди компаний, поставляющих услуги облачных СУБД, можно выделить такие, как Oracle – на основе собственной СУБД [14], Enterprise DB – на основе PostgreSQL [15], Caspio.com – на основе MSSQL [16], ClearDB [17], SkySQL [18] и другие – на основе mySQL. Достаточно подробная классификация сегодняшних облачных систем приведена в работе [19].

Однако все примеры работающих облачных сервисов ориентированы на невысокие требования обслуживания, а главным плюсом этих сервисов

является снижение расходов за счёт отсутствия затрат на владение сервером. При увеличении требований к обслуживанию компании (те же Oracle, Microsoft, Enterprise DB) предлагают владеть собственными серверами.

Таким образом, все перечисленные решения являются коммерческой нишей, а не возможностью увеличивать собственную производительность (в том числе в реальном времени) за счёт использования облака. Несомненно, о чем свидетельствует успешность этих проектов, такая ниша востребована, однако задачу увеличения производительности за счёт облака, в том числе за счёт частного облака (private cloud [11]) или корпоративного (communit□cloud [11]), они не решают.

## 2. Сравнение архитектур СУБД в контексте организации облака

Ключевым фактором при построении облачных систем является выбор архитектуры. На сегодняшний день можно выделить следующие основные используемые архитектуры: shared nothing [20], shared disk [21], архитектуру shared ever□thing, в которой используется программно-реализованный глобальный кеш (Global Cache Fusion) [22] (далее в статье – shared ever□thing), а также решения на основе асинхронной или синхронной репликации, примеры которых будут рассмотрены ниже. Кроме того, несмотря на явную неэффективность с точки зрения производительности, в некоторых случаях используются системы, использующие распределённые транзакции [23], которые в сущности являются частным случаем синхронной репликации. Приведём плюсы и минусы каждой их этих архитектур на примерах существующих реализаций.

**Shared Disk.** Эта архитектура довольно редко используется в СУБД. Среди примеров можно привести Oracle Parallel Server [24], систему, которая уже много лет не поддерживается. Причина довольна проста – для синхронизации данные надо записать на общий диск и прочитать оттуда. Поскольку скорости обращения к дисковым устройствам (время отклика) ниже, чем скорость обращения к памяти, в последующих версиях корпорация Oracle перешла на архитектуру shared ever□thing. Архитектура shared disk согласно [24] также присуща и облачным решениям на основе IBM DB2, хотя использование технологии pureScale [9] переводит эти системы в категорию shared ever□thing.

Стоит отметить, что для систем с небольшим количеством общих ресурсов архитектура shared disk из-за своей простоты может быть полезна. Ещё один недостаток этой архитектуры – низкая производительность файловых систем, корректно работающих с разделяемым несколькими узлами диском, из-за необходимости обеспечения когерентности кешируемых данных.

Резюмируем: shared disk демонстрирует невысокую пригодность к обработке OLTP-трафика.





**Решения на основе репликации**. Решения на основе репликации, как синхронной, так и асинхронной, очень широко распространены. Можно упомянуть standb□ сервера, на которые переносится нагрузка read-onl□ (например, связанная с отчётностью) [25] или так называемая нагрузка read-mostl□, содержащая преимущественно чтение, но, кроме того, очень редко выполняющая запись [25]. Кроме того, стоит упомянуть системы выборочной репликации (например, Oracle Golden Gate [26], Quest SharePlex [27]), а также системы, основанные на применении (часто фильтрованных) бинарных журналов m□SQL. Одной из наиболее удачных систем на основе применения бинарных журналов является Percona XtraDB Cluster от компании Percona [28].

Однако, несмотря на широкое распространение, все перечисленные системы страдают одним недостатком: на каждый узел необходимо реплицировать изменение со всех остальных узлов. Это накладывает жёсткое ограничение: **суммарный объём изменений всех систем не должен превышать возможности самого медленного узла на применение (накат) изменений**. На практике такие системы редко содержат число узлов, большее 3, именно из-за этого ограничения.

Что касается систем, использующих распределённые транзакции (например, [29]), то они являются частным случаем синхронной репликации. Можно было бы много говорить о неэффективности наиболее распространённых алгоритмов 2PL [30] и 2PC [31], но не они являются слабым местом систем, использующих эти алгоритмы (в контексте функционирования PaaS [11]), а сама архитектура на основе репликации. Возможно, именно поэтому модернизированные алгоритмы блокирования и фиксации транзакции [32-34] не находят широкого применения в контексте роста популярности облачных систем.

**Shared nothing.** Примером успешных реализаций этих подходов является m□sql NDB cluster [35], ElasTras [36], Teradata [37], xkoto Gridscale [38]. Стоит отметить, что на этой архитектуре построено наибольшее количество облачных систем. Однако, как будет показано ниже (разд. 3), shared nothing выдвигает требования к OLTP-трафику и только для определённых классов трафика архитектура эффективна (в контексте миграции в облако).

Оптимизация размещения данных по узлам в зависимости от трафика представляет собой серьёзную задачу, решение которой позволит системам, базирующихся на shared nothing соответствовать требованиям PaaS [11]. Всё это требует более подробного анализа проблем масштабирования shared nothing систем в зависимости от трафика.

Оценивая архитектуру **shared everything**, реализованную в системе Oracle Real Application Clusters (Oracle RAC) [39], необходимо констатировать возможность масштабирования системы без явных ограничений на соотношение чтения и записи по сравнению с архитектурой на основе взаимной репликации, несомненное преимущество в быстродействии по

сравнению с архитектурой shared disk и отсутствие зависимости масштабируемости от трафика, присущее системам shared nothing.

Тем не менее, необходимо указать и на ограничения архитектуры shared everthing. В первую очередь – это высокие требования ко времени отклика между узлами. Следствием этого является тот факт, что глобальная сеть в качестве межузлового транспорта может использоваться только при условии существенных ограничений на трафик, которые, как будет показано в этой работе, в конечном счёте накладывают ограничения на интенсивность изменения общих ресурсов. Причём интенсивность изменений ограничена не соотношением изменений в общем трафике при заданном количестве узлов как в архитектуре на основе взаимной репликации, а абсолютная величина интенсивности ограничена непосредственно временем отклика сети. Так же, как и для *shared nothing,* необходим детальный анализ недостатков архитектуры, особенно в сравнении *shared everything* и *shared nothing*.

Отметим также, что система Oracle RAC (взятая в качестве прототипа программно реализованной архитектуры shared everthing) имеет широкий спектр внедрений, обеспечивающий распространённость системы и влияющий на качество её сопровождения, что упрощает исследование, и повышает ценность полученных результатов.

## *3. Основные недостатки архитектуры shared nothing*

Основная проблема в реляционной модели для *shared nothing* – это отсутствие всех необходимых данных на узле, производящем обработку. При этом, как правило, система способна распознавать запросы, для которых все необходимые данные расположены на одном узле и в идеале, если весь трафик будет состоять из таких запросов (назовём их «локализируемые», хотя также может использоваться термин одноузловые [40,41]), то можно говорить об успешном масштабировании. Обработка остальных запросов (назовём их «нелокализируемые»), как правило, реализуется всеми узлами параллельно: т. е. идентичный запрос обрабатывается на каждом узле системы с данными этого узла.

Нелокализируемыми часто становятся запросы, обращающиеся к глобальным индексам, запросы с соединением, а также все запросы, не содержащие ключ разбиения на узлы. Более подробно такие случаи рассмотрены в [1]. Стоит отметить и другой вариант обработки «нелокализованного запроса» в случае общего пространства адресации страниц (далее упоминаемый как «альтернативный»), когда обработкой запроса занимается один узел, запрашивающий недостающие страницы с других узлов.

Сразу же стоит указать, что для хранилищ данных (data warehouse) первая стратегия является доминирующей – при обработке запросом больших объёмов данных, целесообразность вовлечения всех узлов для доступа ко всем данным целесообразна, а для OLTP – нет. Например, компания Teradata чётко позиционирует свою распределённую систему как хранилище данных [37].





Логично предположить, что вторая «альтернативная» непараллельная модель обработки будет более пригодной для OLTP-систем shared nothing, однако практика показывает, что в реализации OLTP-систем shared nothing [41-46] в основном рассматриваются трафики, состоящие только из «локализируемых» запросов, а проработка варианта обработки «нелокализируемых» запросов в них отсутствует.

И действительно бывают трафики, для которых равномерность распределения [45-46] или выбор объектов для шардинга (для минимизации соединений таблиц [41]) определяют эффективность масштабирования. Но тогда нужно точно определить, для каких трафиков это возможно. В случае облачной СУБД, это означает необходимость изучения инженерами облачного провайдера взаимосвязей ресурсов в транзакциях перед тем как получить возможность подтверждения возможности размещения СУБД в облаке, что весьма странно для PaaS[11].

Когда это действительно возможно? Например, в ситуации обновления информации об абоненте или формирования выписки по абоненту, если каждому узлу назначается свой список абонентов, непересекающийся со списками абонентов других узлов, как предлагается в работе [44].

Назовём «полностью локализируемым» трафик, обладающий такими свойствами: все запросы включают в себя идентификатор сущности и являются локализируемыми, критерием локализации (назначения узла обработчика) выступает идентификатор сущности, указываемый в запросе. Для примера [44] уже поисковый запрос с фильтром по дате оплаты, трансформируется в $n$ запросов ко всем узлам, где $n$ – количество узлов. С обработкой трафика, состоящего только из «локализируемых» запросов, связана и популярность NoSQL в контексте масштабирования в облаке. Так же, как и случай, рассмотренный в [44], по большей части хорошо масштабируются в облачных системах и трафики социальных сетей. Заметим, что «полностью локализируемый» трафик всегда масштабируем в облаке.

И здесь необходимо обратить внимание на очень интересный факт, меняющий в контексте изучения критериев трансформации СУБД в облачную местами причину и следствие: NoSQL-СУБД могут эффективно обслуживать только «полностью локализируемый» трафик, а значит, базы данных таких систем легко автоматически разбивать на партиции по первичному ключу. Т.е. задача помещения таких задач в облако, удовлетворяющее условиям PaaS, – тривиальна. Это мы и наблюдаем на практике.

С инженеров облачного провайдера автоматически снимается задача анализа пригодности трафика для облака, организованного на архитектуре shared nothing. Критерием пригодности размещения в облаке становится использование NoSQL-СУБД. Выбор разработчиком NoSQL при проектировании приложения автоматически выполняет работу за инженера облачного провайдера.

Нужно отметить и тот факт, что непосредственно на выборку и модификацию данных при обработке OLTP запросов тратиться намного меньше времени, чем на сумму обслуживания структур управления общим ресурсом, сетевого взаимодействия, целостности, безопасности, разбора (parsing) и других задержек запроса. Во многом такое распределение задержек стало причиной популяризации NoSQL-СУБД, хотя наиболее значащим фактором является эффективность масштабирования и финансовая доступность. Во всяком случае, впечатляющих результатов сравнения одноузловых систем MongoDB (или другой NoSQL-СУБД) и Oracle в пользу первой системы автору этой статьи наблюдать не доводилось.

Возвращаясь к «альтернативному» подходу при обработке нелокализируемых запросов трафика, отметим ключевое значение индексов для OLTP-систем, что приводит к малому количеству страниц, используемых каждым запросом, что указывает на перспективность такого подхода. Однако отсутствие реализаций этого подхода и выбор традиционного подхода (плодящего «нелокализируемые» запросы на каждом узле) заставляет серьёзно задуматься о причинах такой ситуации.

Очевидно, что «альтернативный» вариант обработки нелокализируемых данных требует закладывания возможности глобальной адресации страниц внутри облачной системы shared nothing на уровне самой СУБД, а современные реализации архитектуры shared nothing часто проектируются в расчете на одноузловое использование без поддержки такой возможности. Однако, что будет показано ниже, альтернативный вариант может уступать в эффективности и архитектуре shared everything.

Резюмируя недостатки shared nothing, необходимо констатировать неэффективность обработки нелокализируемых запросов в OLTP-системах, что связано с отсутствием проработки «альтернативного» варианта и архитектурной неэффективностью классического подхода. Примеры успешных реализаций кластеров shared nothing при обработке OLTP-трафика **всегда содержат ключ разбиения на партиции в условиях поиска**, что далеко не всегда присуще OLTP-трафику. Можно говорить о невысокой эффективности использования подобных систем в качестве обработчиков OLTP-трафика, если опираться на существующие реализации этой архитектуры.

## *4. Проблема нарастающей очереди как ключевой недостаток архитектуры shared everything*

Для архитектуры *shared everything* [22] ограничением масштабируемости может стать проблема нарастающей очереди [47]. Суть проблемы состоит в том, что при превышении некоторого уровня интенсивности запросов система теряет работоспособность. И эта интенсивность зависит от среднего времени блокирования ресурсов, входящих в транзакции трафика, и взаимосвязей транзакций и ресурсов [1].





Следствием является то, что наращивание количества узлов не позволяет увеличить интенсивность трафика при сохранении его характера. Т.е. масштабируемость ограничена некоторым уровнем интенсивности, который обратно пропорционалена среднему времени обработки «критически горячего» класса транзакций. «Критически горячим» в данном случае называется класс транзакций, для которого при повышении интенсивности трафика величина, обратная показателю интенсивности выполнения транзакций этого класса, превышает среднее время выполнения транзакций этого класса. Рассмотрим этот эффект подробнее.

Для начала определимся с терминологией. Транзакции могут быть разных классов. Будем считать транзакции принадлежащими к одному классу, если они конкурируют за один и тот же набор классов ресурсов, соблюдают один и тот же порядок обработки классов ресурсов, а также поддерживают один и тот же режим выполнения (чтение или изменение). В свою очередь, классом ресурсов будем называть ресурсы, относящиеся к одинаковым логическим сущностям.

Поясним эти громоздкие определений на примере общеизвестного трафика TPC-C. Здесь классы транзакций – это New-Order, Payment, Order-Status, Delivery, Stock-Level, а классы ресурсов – это строки соответствующих таблиц customer, warehouse, district, orders, order_line и т.д. Кроме того, классами ресурсов являются входы индексов на соответствующие таблицы.

Предположим, что мы имеем систему, на которую поступают заявки с интенсивностью *v*. Пусть в каждой заявке содержится одна транзакция какого-либо класса. У системы имеется множество общих ресурсов. Обработка заявки занимает время *t*. Если заявка, которая поступила в систему, обратившись к общему ресурсу, обнаруживает, что этот ресурс заблокирован, она ожидает его освобождения. Если более одной заявки ожидает общего ресурса, доступ к ресурсу осуществляется в порядке очереди FIFO.

Пусть время обработки транзакции некоего класса равно *t*. Для 1-й заявки время выполнения равно *t*. Если одновременно в обработке две заявки, то время обработки второй заявки – это *t* плюс время ожидания окончания обработки текущей заявки $t_x$. Заметим, что $t_x$ меньше *t*, т.к. вторая заявка поступила после первой. Тогда, если в обработке три заявки (очередь из двух заявок), то время обработки 3-й заявки – $t_x+2t$. Таким образом, при очереди из *n* заявок,время обработки $(n+1)$-ой заявки составит $t_x+nt$.

Несложно показать, что при превышении показателя интенсивности $1/t$ очереди в системе будут нарастать, и система уйдёт в так называемую перегрузку, т. е. перестанет обрабатывать заявки. Более интересным свойством будет и то, что при образовавшейся во время локального пика очереди интенсивность перехода в перегрузку будет значительно ниже $1/t$.

Важно отметить, что ожидание заблокированных ресурсов увеличивают время обработки заявки, а увеличение времени обработки заявки, в свою очередь, увеличивает вероятность блокирования ресурса. Кроме того, для классов

транзакций, которые пересекаются по ресурсам, увеличение времени обработки одного из классов ресурсов может привести к перегрузке. Всё это, несмотря на простоту рассмотренных ситуаций и очевидность выводов, создаёт серьёзные проблемы для программной реализации глобального кеша в реализации архитектуры shared everything в Oracle RAC [22].

Выход из этого положения один – сокращать время обработки заявки. Однако стратегий реализации при страничной организации глобального кеша несколько –: сокращение времени доступа к странице, сокращение времени блокирования ресурсов транзакцией (в том числе за счёт распараллеливания выборки страниц необходимых для транзакции [1]) и сокращение длительности обработки ресурсов транзакцией.

Заметим, что компания Oracle решает проблему первым способом, уменьшая время отклика межузлового канала на основе технологии InfiniBand [48]. Кроме того, с учетом существующих мощностей процессоров и относительной быстроты процессорной обработки OLTP-транзакций по сравнению с сетевыми задержками при этой обработке нецелесообразным является сокращение длительности обработки ресурсов транзакцией. В данной статье мы ограничимся первой стратегией, однако, в отличии от подхода Oracle, сосредоточимся на сокращении среднего времени доступа к странице путем изменения способов доступа.

## 5. *Основное преимущество архитектуры shared everything*

Продолжим сравнение архитектур shared everything и shared nothing. Поскольку для OLTP-систем «альтернативный» подход выигрышнее, сравнение будем проводить между программной реализацией архитектуры shared everything, применяемой в Oracle RAC, и «альтернативным» вариантом shared nothing. Если рассматривать в качестве критерия эффективности предельную интенсивность, при которой сохраняется способность обрабатывать заявки определённого трафика, то преимущество shared everything можно достичь за счёт сокращения среднего времени доступа к странице. Это обеспечивается кешированием страниц, но требует более сложного механизма обеспечения когерентности. Более подробно алгоритм обработки страниц описан в [1]; ниже приводится его краткое описание.

Ключевой идеей является назначение каждой странице узла мастера, управляющего состояниями копий страницы в локальных кешах каждого узла. Выделяются такие состояния (называемые также блокировками): XCUR (xc, exclusive current) – монопольное текущее, SCUR (sc, shared current) – совместное текущее, CR (consistent read) – целостное чтение. Первое состояние необходимо для внесения изменений, второе используется для чтения последней версии страницы, а третье также используется при чтении и обеспечивает то, что версия страницы соответствует некоторому моменту в прошлом.





Стоит отметить, что реализация в Oracle мультиверсионности за счёт версионности страниц является плюсом при организации многоузловой системы, поскольку CR-копии страниц не требуют синхронизации между узлами. Как следствие мастер-узел отслеживает преимущественно первые два состояния (xc и sc). XCUR-блокировкой в каждый момент времени может владеть только один узел. Блокировкой же SCUR может владеть сколько угодно узлов. Снятие со страницы XCUR-блокировки (переход в SCUR или XCUR на другом узле) приводит к переводу состояния страницы на отдающем узле в CR (через промежуточное состояние PI), а снятие SCUR – к переводу состояния всех страниц, где был SCUR, также в CR. Отметим существование промежуточного состояние PI, необходимого для гарантирования целостности, но к рассматриваемым вопросам сравнения эффективности архитектур не относящееся.

Сама последовательность пересылок, реализованная в Oracle для обеспечения когерентности, достаточно проста: при доступе к странице сначала идёт обращение к мастер-узлу с запросом необходимой блокировки (на страницу XCUR или SCUR), затем мастер шлёт узлу, владеющему в данный момент блокировкой, инструкцию переслать текущую версию страницы запросившему узлу, после чего происходит пересылка самой страницы. После этого узел может начать обработку страницы. Параллельно с этим идёт оповещение мастер-узла о получении страницы (и соответственно блокировки xc или sc на страницу). Таким образом, для доступа к странице мы вынуждены выполнить три пересылки.

Здесь важно отметить два фактора, влияющих на количество пересылок: наличие в кеше узла, выполняющего обработку нужной версии страницы с требуемой блокировкой, и то, что узел, обрабатывающий страницу, оказывается мастером для неё. В обоих случаях уменьшается количество пересылок, и, как следствие, сокращается среднее время доступа к странице.

Необходимо обратить внимание на замечательный факт: для статических данных время доступа к странице сокращается существенно, поскольку через некоторое время все активно используемые страницы на каждом узле будут закешированы (получат блокировку SCUR) всеми узлами. Это является главным плюсом архитектуры, реализованной в Oracle RAC, по сравнению с другими архитектурами.

Но стоит сказать и о главном минусе по сравнению с «альтернативным» подходом архитектуры shared nothing (в первую очередь): три пересылки до начала обработки страницы против двух (без учёта ожидания очереди к странице) в случае незакешированных данных. (В работе [1] показана второстепенность фактора мастер-узла по сравнению с кешированием, поэтому сосредоточимся на эффекте кеширования.) При сравнении, если полагать отсутствие очередей, получается интересная ситуация: если при обращении к горячим страницам одного класса преобладает чтение, то лучшее

среднее время обращения показывает стратегия RAC, а при преобладании операций записи «альтернативный подход».

Но нужно отметить и то, что для именно для «горячих» страниц с увеличением интенсивности будут образовываться очереди. Как будет показано ниже, именно при их образовании создаётся описанный выше эффект нарастающей очереди и видна перспективность подхода, применяемого Oracle. Кроме того, если преобладают операции записи в горячую страницу (что легко отследить на при формировании очереди запросов к мастер-узлу), выполняется простые, но эффективные действия: узлу, запросившему блокировку SCUR, узел, владеющий блокировкой XCUR, не удовлетворяя SCUR, отдаёт CR-версию на самый последний (current) момент времени. Таким образом, интенсивность заявок, приводящая к созданию очереди, сокращается до интенсивности записи в горячую страницу.

Для демонстрации эффективности shared ever□thing при возникновении очередей рассмотрим возможную реализацию когерентности при «альтернативном» подходе. Напомним, что после изменения страницы её нужно вернуть узлу-владельцу. И возникает дилемма: или при одновременном запросе страницы несколькими узлами только один из них сможет вносить изменения, или узел владелец распараллелит изменения в страницы, а при получении их будет выступать арбитром?

Второй подход требует глубокой проработки и рамках этой статьи рассматриваться не будет, а при применении первого подхода мы наблюдаем явное преимущество подхода Oracle, поскольку шаг очереди увеличивается в два раза: сначала пересылка владельцу страницы изменённой копии, а потом её же пересылка следующему в очереди узлу. Обратим внимание, что если узлам дать возможность пересылать страницу друг другу напрямую, то мы получим как раз подход, применяемый Oracle, плюс третью пересылку (от обрабатывающего узла мастеру на фоне обработки страницы) для контроля того, где находится текущая версия страницы.

Следствием увеличения шага очереди в два раза является уменьшение уровня интенсивности, приводящего к переходу в перегрузку, также примерно в два раза. Таким образом, получаем весьма интересные выводы: плюсом архитектуры shared ever□thing, реализованной в RAC, является лучшая устойчивость к перегрузкам при доступе к горячим страницам, а минусом – большее (до полутора раз) время доступа при обращении к таким «холодным» страницам, для которых запись преобладает над чтением (холодным настолько, что к ним практически не образуется очередь).

Обратим внимание и на тот факт, что при увеличении количества узлов системы при том же соотношении числа операций чтения и записи эффективность применения кеша снижается, что также говорит о том, что при доступе к «холодным страницам» альтернативный подход shared nothing лучше. Поэтому актуальна задача разработки такого способа доступа к





страницам, который будет обладать полезными свойствами как первого, так и второго подходов и будет эффективно решать проблему нарастающей очереди.

## *6. Метод назначения обработчика ресурса*

Для повышения эффективности требуется решить задачу распределения между входящими в систему узлами всех имеющихся во всех очередях транзакций и последовательности обработки страниц при условии, что целевой функцией качества размещения является снижение времени пересылок между узлами при сохранении требований ACID. Поскольку эту задачу приходится решать в реальном времени, необходимо использовать простые и эффективные алгоритмы планирования.

Очевидно, что наилучший вариант получится, если все транзакции, связанные в очередь, выполнять на одном узле. Но тогда мы упрёмся в ограничения масштабируемости. Попробуем для горячих страниц локализовать обработку на одном узле. Примером реализации локализации, является метод разгрузки очереди, описанный в [49], с изменениями, описываемыми ниже в данной статье. Результатом применения метода является «разгрузка», устранение очереди к распределённой странице памяти, очереди, возникшей в результате того, что уровень интенсивность к «горячей» странице кратковременно или долговременно превысила граничный уровень интенсивности «перегрузки».

Идея метода достаточно проста: если нарастает очередь, то необходимо снизить количество пересылок между узлами. Для этого выбирается узел, который будет монопольно вносить изменения в «горячую» страницу (в дальнейшем – хостер). При этом на чтение страница (в состоянии CR) отдаётся каждому запросившему её узлу.

Возникает вопрос, как именно хостеру будут передаваться изменения, которые необходимо внести в страницу. Первый вариант простой: узел, не являющийся хостером страницы (в дальнейшем инициатор), сначала нужную ему страницу читает (запрашивает у хостера в состоянии CR на самый последний момент времени), затем формирует запись об изменениях, которые нужно внести в страницу (аналогично тому, как это делается для журнала, обеспечивающего долговечность (durabilit□) ACID-транзакций), и эту запись пересылает хостеру вместе со старыми данными для внесения в страницу. Хостер в случае соответствия старых и новых данных (на уровне строки) вносит изменения и возвращает получившуюся страницу инициатору.

Недостаток такого подхода в том, что в случае «горячей» строки может возникать большое количество исключений, на обработку каждого из которых потребуется две дополнительные пересылки; более того, эти пересылки будут увеличивать ожидание в очереди.

Второй вариант более интересен, но и более сложен: узлу хостеру отправляется кусок кода SQL-запроса выполняющего все действия над этой страницей (возможно, и читающий другие страницы; назовём его SQL-bit).

Первой сложностью является то, что операции изменения транзакции выполняются в разных узлах. Эта сложность обходится, если изменения рассматриваемой транзакции, выполняемые хостером, ещё раз журнализовать в узле-инициаторе. Для обеспечения большей надёжности это может быть и преимуществом.

Вторая сложность – это прерывание выполнения SQL-запроса на инициаторе вызовом удалённой процедуры обработки горячей страницы на хостере. Сложность здесь чисто техническая: требуется интеграция интерпретатора SQL с механизмом распределённой адресации (и кеширования) страниц.

Изложим возможную структуру обмена сообщениями для реализации второго варианта упомянутого метода.

1. При возникновении длинной очереди мастер-узел должен перевести страницу в режим разгрузки, т.е. в некоторый момент времени (когда образовалась очередь) узлы, ожидающие очереди, должны быть уведомлены о переводе страницы в режим разгрузки, и о том, какой узел назначен хостером. Это выдвигает требования к мастер-узлу отслеживать состояние очередей к страницам и выбирать момент для включения режима разгрузки. Кроме того, мастер должен уметь отключать состояние разгрузки для страницы и оповещать соответствующие узлы о таком изменении.
2. На запросы страниц другими инициаторами мастер-узел отвечает, что страницы находятся в состоянии разгрузки с указанием узла-хостера, которому надо пересылать SQL-bit. Инициаторы запоминают, какие страницы находятся в состоянии разгрузки, чтобы в дальнейшем обращаться к ним, минуя мастера.
3. Инициатор формирует и пересылает хостеру свой SQL-bit.
4. Хостер вносит изменения в страницу и в журнал упреждающей записи, а затем пересылает инициатору текущую версию страницы в состоянии CR и свою журнальную запись об изменениях.
5. Инициатор получает страницу в состоянии «как будто он сам внёс в неё изменения и сразу же отправил дальше по запросам других узлов», и заносит журнальную запись хостера в собственный журнал упреждающей записи.

Принципиально заложить в алгоритм возможность мигрировать с узла, вносящего изменения, на другой узел для обеспечения динамической балансировки нагрузки в случае наличия нескольких горячих ресурсов транзакции. т.е. рассылку мастером сообщений о переназначении хостера.

Условимся говорить, что страница находится в состоянии «разгрузки», если к ней начал применяться описанного алгоритм. Выходом из «разгрузки» условимся называть процесс, по результатам которого прекращается применение описанного алгоритма и снова начинает применяться классический способ доступа к странице, применяемый в Oracle RAC.





Важным нюансом в этом алгоритме является минимизация ресурсов на мониторинг возникновения очереди. Кроме того, выход из состояния «разгрузки», оценка целесообразности выхода и приоритизация назначения узла, выполняющего «разгрузку», т.е. всё то, что относится к переходным процессам, связанным с «разгрузкой», выходят за рамки данного исследования.

## *7. Сокращение количества пересылок при доступе к странице для страниц SCUR mostly*

Замечательным свойством мастер-узла при использовании способа доступа, применяемого в Oracle RAC, является возможность отслеживания статистики нахождения в состояниях SCUR, XCUR и «разгрузки». Для страниц SCUR mostl□, т.е. таких, которые в основном читаются, но иногда меняются, имеет смысл при удовлетворении блокировки SCUR после XCUR пересылать еще одну копию SCUR-страницы мастер-узлу.

Тогда при последующих SCUR-запросах (т.е. запросах на чтение страницы) вместо трёх мы будем выполнять две пересылки, как и при «альтернативном» подходе shared nothing. Это простое решение позволит сократить среднее время доступа к странице, однако важно отметить, что при преобладании записи над чтением дополнительная пересылка мастер-узлу становится нецелесообразной.

## *8. Заключение*

В заключение статьи обратим внимание на то, что очереди за страницы не являются единственным источником конфликтов в OLTP-системах. Во многих трафиках основной причиной перегрузки является конфликт за ресурсы одного класса, располагаемые на страницах. Основное отличие состоит в том, что блокировка ресурса (на примере Oracle RAC) обеспечивается до завершения транзакции, а блокировка страницы (и, как следствие, отправка её следующему в очереди узлу) прекращается после внесения изменений.

Несмотря на взаимосвязанность задержек, порождаемых этими конфликтами, для каждого из них имеется своя стратегию разрешения, и данная статья вообще не касается способов организации блокировкия ресурсов и разрешения конфликтов транзакций. Это безусловно является актуальной темой для исследований, и идеи работы [1] в этом направлении необходимо развивать.

Также стоит отметить, что рассматриваемый в статье «альтернативный подход», можно классифицировать не как разновидность *shared nothing*, а как реализацию *shared everything*, отличную от применяемой в Oracle RAC реализации общей памяти на основе Global Cache Fusion (программно реализованного глобального кеша).

Подчеркнем важную роль третьего состояния страницы (разгрузка) для обеспечения страничной организации в любых распределённых системах, а не

только в СУБД. Наконец, еще раз заметим, что эффект от применения разгрузки зависит от трафика, и наибольший эффект происходит в случае коротких транзакций трафика.

## Список литературы


[1]. Гусев Е.И. Способы организации совместного доступа к распределённым страницам памяти в системах облачных вычислений. Диссертация на соискание учёной степени кандидата технических наук. НТУ "КПИ" им. Сикорского, Киев, 2017. 156 страниц.

[2]. Кузнецов С.Д., Посконин А.В. Системы управления данными категории NoSQL. Программирование, том 40, № 6, 2014, стр. 34-47

[3]. Berndt D.J., Lasa R., McCart J. SiteWit Corporation: SQL or NoSQL? That is the Question!. Journal of Information Technolog□ Education: Discussion Cases, Volume 6, 2017, pp. 04 Universit□of South Florida, 2012, p. 14-15. DOI:10.28945/3920.

[4]. Бурмистров А.В., Белов Ю.С. Недостатки реляционных баз данных. Электронный журнал: наука, техника и образование, 2015, № 3. стр. 25-34.

[5]. Селезнев К. От SQL к NoSQL и обратно. Открытые системы. СУБД, 2012. No 2. Доступно по ссылке: https://www.osp.ru/os/2012/02/13014127/, 10.01.2018.

[6]. Padh□ R.P., Patra R.M., Satapath□ S.C. RDBMS to NoSQL: Reviewing Some Next-Generation Non-Relational Database's. International Journal of Advanced Engineering Sciences and Technologies, 2011, vol. 11 (1), pp. 15-30.

[7]. Мухина Ю. Р. Обзор NoSQL решений управления данными. Управление в современных системах. 2013. No 1. стр.68-73.

[8]. Weiss R. Technical Overview of the Oracle Exadata Database Machine and Exadata Storage Server. ORACLE white paper, June 2012. Доступно по ссылке: http://www.oracle.com/technetwork/database/exadata/exadata-technical-whitepaper-134575.pdf, 10.01.2018.

[9]. IBM Documentation: DB2 pureScale Feature road map (online). Доступно по ссылке: https://www.ibm.com/support/knowledgecenter/SSEPGG_11.1.0/com.ibm.db2.luw.licensing.doc/doc/c0056030.html, 10.01.2018.

[10]. Rackspace Support "Understanding the Cloud Computing Stack: SaaS, PaaS, IaaS," Rackspace, October 22, 2013. Доступно по ссылке: https://support.rackspace.com/how-to/understanding-the-cloud-computing-stack-saas-paas-iaas/ , 10.01.2018.

[11]. Mell P., Grance T. The NIST Definition of Cloud Computing. National Institute of Science and Technolog□ Special Publication 800-145,October 25, 2011. Доступо по ссылке https://www.nist.gov/sites/default/files/documents/itl/cloud/cloud-def-v15.pdf, 10.01.2018.

[12]. Butler B. PaaS Primer: What is platform as a service and wh□does it matter?. Network World, Februar□11, 2013. Доступно по ссылке: https://www.networkworld.com/article/2163430/cloud-computing/paas-primer-what-is-platform-as-a-service-and-wh□-does-it-matter-.html, 10.01.2018.

[13]. Kadam M, Jidge P., Tambe S., Ta□ade E. Cloud Service Based On Database Management S□stem , Int. Journal of Engineering Research and Applications. ISSN:2248-9622, Vol. 4, Issue 1( Version 3), Januar□2014, pp. 303-306.

[14]. Oracle Infrastructure and Platform Cloud Services Securit□ Oracle white paper, November 2016. Доступно по ссылке:







https://cloud.oracle.com/opc/iaas/whitepapers/Oracle_Cloud_Securit□_Whitepaper.pdf, 10.01.2018.

[15]. Achieving HIPAA compliance with Postgers Plus Cloud Database. EnterpriseDB white paper. EnterpriseDB Corporation, 2015. https://www.enterprisedb.com/hipaa-compliance-postgres-plus-cloud-database 10.01.2018.

[16]. Online Database Software. Custom Database Applications. Caspio. Доступно по ссылке: https://www.caspio.com/, 10.01.2018.

[17]. ClearDB - The Ultra Reliable, Geo Distributed Data Services Platform. Доступно по ссылке: http://w2.cleardb.net/ , 10.01.2018.

[18]. Sk□SQL Makes Highl□Available Databases Eas□, with MariaDB Enterprise | MariaDB Доступно по ссылке: https://mariadb.com/about-us/newsroom/press-releases/sk□sql-makes-highl□-available-databases-eas□-mariadb-enterprise, 10.01.2018.

[19]. Николаенко А. Год облачных СУБД. Открытые системы.СУБД, 2013, No 9. Доступно по ссылке: https://www.osp.ru/os/2013/09/13038286/ , 10.01.2018.

[20]. Shared disk architecture – Wikipedia. Доступно по ссылке: https://en.wikipedia.org/wiki/Shared_disk_architecture, 10.01.2018.

[21]. Shared-nothing architecture – Wikipedia. Доступно по ссылке: https://en.wikipedia.org/wiki/Shared_nothing_architecture, 10.01.2018.

[22]. Parallel Execution with Oracle Database 12c Fundamentals. Oracle White Paper, December 2014. Доступно по ссылке http://www.oracle.com/technetwork/database/bi-datawarehousing/twp-parallel-execution-fundamentals-133639.pdf, 10.10.2018.

[23]. Taniar D., Leung C. H. C., Raha□u W., Goel S. High Performance Parallel Database Processing and Grid Databases. Ch. 10. Wile□ Publishing, 2008, isbn: 9780470107621, pp. 289-320.

[24]. Bauer M. Oracle8i Parallel Server Concepts, Release 2 (8.1.6), Part No. A76968-01, December 1999. Доступно по ссылке: https://docs.oracle.com/cd/A87860_01/doc/server.817/a76965.pdf, 10.01.2018.

[25]. Oracle Active Data Guard, Real-Time Data Protection and availabilit□ Oracle White Paper, October 2015. Доступно по ссылке: http://www.oracle.com/technetwork/database/availabilit□/active-data-guard-wp-12c-1896127.pdf, 10.01.2018.

[26]. Oracle GoldenGate 12c: Real-Time Access to Real-Time Information. Oracle White Paper, March 2015. Доступно по ссылке: http://www.oracle.com/us/products/middleware/data-integration/oracle-goldengate-realtime-access-2031152.pdf, 10.01.2018.

[27]. Chu T. Top Five Reasons to Choose SharePlex® Over Oracle GoldenGate. Quest Software, November, 2011. Доступно по ссылке: http://www.dlt.com/sites/default/files/Quest-Shareplex-Whitepaper.pdf, 10.01.2018.

[28]. Percona XtraDB Cluster Release5.7.17-29.20 Operations Manual Доступно по ссылке: https://learn.percona.com/download-percona-xtradb-cluster-5-7-manual, 10.01.2018.

[29]. Xiai Yan, Jinmin Yang, Qiang Fan. An Improved Two-phase Commit Protocol Adapted to the Distributed Real-time Transactions. Przegląd Elektrotechniczn□ (Electrical Review), ISSN 0033-2097, R. 88 NR 5b/2012, pp. 27-30.

[30]. Bernstein P. A., Hadzilacos Goodman N.: Concurrenc□ Control and Recover□ in Database S□stems, Addison Wesle□ Publishing Compan□, ISBN 0-201-10715-5. 1987 pp. 49-53.

[31]. Open Group Standard DRDA, Version 5, Volume 3: Distributed Data Management (DDM) Architecture // ISBN: 1-931624-93-3 Document Number: C114. pp. 831-832.

[32]. Gra□J., Lamport L. Consensus on Transaction Commit. Microsoft Research. 1 Januar□ 2004 revised 19 April 2004, 8 September 2005. Доступно по ссылке: https://www.microsoft.com/en-us/research/publication/consensus-on-transaction-commit/, 10.01.2018.

[33]. Mahmoud H. A., Arora V., Nawab F., Agrawal D., El Abbadi A. Maat: Effective and scalable coordination of distributed transactions in the cloud. Proceedings of the VLDB Endowment, Volume 7, No 5. Januar□2014, pp. 329–340.

[34]. Keidar I. Dolev D. Increasing the Resilience of Distributed and Replicated Database S□stems.Journal of Computer and S□stem Sciences (JCSS). December 1998, Issue 57 (3), pp. 309–324. DOI:10.1006/jcss.1998.1566

[35]. M□SQL :: M□SQL 5.7 Reference Manual :: 21 M□SQL NDB Cluster 7.5 and NDB Cluster 7.6. Доступно по ссылке: https://dev.m□sql.com/doc/refman/5.7/en/m□sql-cluster.html, 10.01.2018.

[36]. Das S., Agarwal S., Agrawal D., El Abbadi A. ElasTraS: An Elastic, Scalable, and Self Managing Transactional Database for the Cloud. UCSB Computer Science Technical Report 2010-04, pp. 1-14.

[37]. The Teradata Scalabilit□Stor□, A Teradata White Paper, EB-3031 0701, 2001, NCR Corporation. Available at: http://www3.cs.ston□brook.edu/~sas/courses/cse532/fall01/teradata.pdf, 10.01.2018.

[38]. Gridscale® Database Virtualization Software. Technical whitepaper, xkoto, Inc. Item: GS-WP-EN-20080930. 2008. Доступно по ссылке: http://www.tech-21.com.hk/download/Gridscale_Technical_White_Paper.pdf , 10.01.2018.

[39]. Michalewicz M., Clouse B., McHugh J. Oracle Real Application Clusters (RAC). Oracle White Paper. June 2013. Доступно по ссылке: http://www.oracle.com/technetwork/database/options/clustering/rac-wp-12c-1896129.pdf, 10.01.2018.

[40]. Кузнецов С.Д. Транзакционные параллельные СУБД: новая волна. Труды Института системного программирования, т. 20, М., ИСП РАН, 2011, стр. 189-251

[41]. Stonebraker M., Madden S., Abadi D.J., Harizopoulos S., Hachem N., Helland P. The End of an Architectural Era (It's Time for a Complete Rewrite). Proceedings of VLDB, 2007, Vienna, Austria, pp. 1150-1160.

[42]. Бойченко А.В., Рогожин Д. К., Корнеев Д. Г. Алгоритм динамического масштабирования реляционных баз данных в облачных средах. Экономика, Статистика и Информатика No 6 (2), 2014. стр. 461-465.

[43]. Чистов В.А., Лукьянченко А.В. Автоматизация масштабирования высоконагруженных баз данных M□SQL. Современные наукоёмкие технологии, 2016, 6-2, стр. 315-319.

[44]. Горобец В.В. Математические модели и алгоритмы оптимизации размещенения данных транзакционных систем. Диссертация на соискание ученой степени кандидата технических наук. Новочеркасск, Южно-Российский государственный политехнический университет (НПИ) имени М. И. Платова, 2015. 210 страниц.

[45]. Зернов А.С., Ожиганов А.А. Горизонтальное масштабирование базы данных с использованием консистентного хеширования. Известия высших учебных заведений. Приборостроение. 2017. Т. 60. № 3. стр. 234-238.

[46]. Caio H. Costa, João Vianne□, Paulo Maia, Francisco Carlos M. B. Oliveira. Sharding b□ Hash Partitioning - A Database Scalabilit□Pattern to Achieve Evenl□Sharded Database Clusters. 17th International Conference on Enterprise Information S□stems (ICEIS 2015), At Barcelona, Spain. DOI: 10.5220/0005376203130320.





[47]. Гусев Е.И. Исследование области применения неблокирующего алгоритма фиксации распределённых транзакций. Вісник НТУУ "КПІ". Сер. Інформатика, управління та обчислювальна техніка. 2012. Випуск 57. стр. 76-80.

[48]. InfiniBand – Wikipedia. Доступно по ссылке: https://en.wikipedia.org/wiki/InfiniBand, 10.01.2018

[49]. Гусев Е.И. Оптимизация доступа к распределённым страницам памяти в cloud computing системах основанных на shared ever□thing архитектуре используя метод разгрузки очередей. Проблеми інформатизації та управління. 2015. – Том 4, № 52. стр. 17-21.

# Optimizing access to memory pages in software-implemented global page cache systems.

*E.I. Gusev <eugene@levelextra.ru>*
*National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute",*
*37, Prosp. Peremohy, Kyiv, Ukraine, 03056.*

**Abstract.** This article is based on a thesis “Techniques of organizations shared access to distributed memor□ pages in cloud computing s□stems”, defensed in Igor Sikorsk□ K□iv Pol□technic Institute in 2017. The paper describes distributed pages processing in Oracle Real Application Clusters (Oracle RAC) and compares it with other known processing methods. The comparison comprises anal□sis of different architectures (including shared nothing, shared disk and “based on a replication” architecture) in the context of SQL quer□ processing and asserts reasonableness of distributed pages approach (also known as Global Cache Fusion) choice for cloud DBMS. As a result researching the Global Cache Fusion approach there was revealed main drawback of Oracle RAC s□stems – “increasing queue problem”: impossibilit□process requests after intensit□of requests exceeds threshold intensit□, which is inversel□ proportional packet sent time between nodes. To eliminate “increasing queue problem” during distributed page access the new access method is proposed, which is based on the initiation of one more page state - the "unloading" state, which increases the efficienc□ of processing distributed pages b□reducing the number of transfers between nodes during hot page processing. The considered method can be used not onl□in cloud DBMS but also in other cloud s□stems in a case if the□use page-organized distributed memor□architecture.

**Keywords:** Oracle RAC; distributed page; shared nothing; shared ever□thing; Global Cache Fusion; RDBMS.

**DOI:** 10.15514/ISPRAS-2018-30(1)-11

**For citation:** Gusev E.I. Optimizing access to memor□pages in software-implemented global page cache s□stems. Trud□ISP RAN/Proc. ISP RAS, vol. 30, issue 1, 2018, pp. 161-182 (in Russian). DOI: 10.15514/ISPRAS-2018-30(1)-11

## References


[1]. Gusev E.I. Techniques of organizations shared access to distributed memor□ pages in cloud computing s□stems. Thesis for a Candidate of Technical Sciences degree, National Technical Universit□of Ukraine "Igor Sikorsk□K□iv Pol□technic Institute", Kiev, 2017. 156 pages.

[2]. Kuznetsov S. D., Poskonin A. V. NoSQL data management s□stems. Programming and Computer Software. November 2014, Volume 40, Issue 6, pp. 323-332. DOI: 10.1134/S0361768814060152.

[3]. Berndt D.J., Lasa R., McCart J. SiteWit Corporation: SQL or NoSQL? That is the Question!. Journal of Information Technolog□Education: Discussion Cases, Volume 6, 2017, pp. 04 Universit□of South Florida, 2012, p. 14-15. DOI:10.28945/3920.

[4]. Burmistrov A.V. Belov Y.S. Disadvantages of relational databases. [Online journal: Science, technolog□ and education]. Elektronn□ zhurnal: Nauka, tekhnika i obrazovanie]. ISSN 2312-8267(Print), ISSN 2413-5801(Online), 2015 No. 3. pp. 25-34 (in Russian).

[5]. Seleznev K. From SQL to NoSQL and back again. Open S□stems. DBMS, ISSN 1028-7493, 2012. No 2 (in Russian). Available at: https://www.osp.ru/os/2012/02/13014127/, 10.01.2018.

[6]. Padh□R.P., Patra R.M., Satapath□S.C. RDBMS to NoSQL: Reviewing Some Next-Generation Non-Relational Database's. International Journal of Advanced Engineering Sciences and Technologies, 2011, vol. 11 (1), pp. 15-30.

[7]. Mukhina Y.R. NoSQL solutions of data management rewiew. Upravlenie v sovremenn□h sistemah, ISSN 2311-1313, 2013. No 1. pp. 68-73 (in Russian).

[8]. Weiss R. Technical Overview of the Oracle Exadata Database Machine and Exadata Storage Server. ORACLE white paper, June 2012. Available at: http://www.oracle.com/technetwork/database/exadata/exadata-technical-whitepaper-134575.pdf, 10.01.2018.

[9]. IBM Documentation: DB2 pureScale Feature road map (online). Available at: https://www.ibm.com/support/knowledgecenter/SSEPGG_11.1.0/com.ibm.db2.luw.licensing.doc/doc/c0056030.html, 10.01.2018.

[10]. Rackspace Support "Understanding the Cloud Computing Stack: SaaS, PaaS, IaaS," Rackspace, October 22, 2013. Available at: https://support.rackspace.com/how-to/understanding-the-cloud-computing-stack-saas-paas-iaas/ , 10.01.2018.

[11]. Mell P., Grance T. The NIST Definition of Cloud Computing. National Institute of Science and Technolog□ Special Publication 800-145,October 25, 2011. Доступо по ссылке https://www.nist.gov/sites/default/files/documents/itl/cloud/cloud-def-v15.pdf, 10.01.2018.

[12]. Butler B. PaaS Primer: What is platform as a service and wh□does it matter?. Network World, Februar□11, 2013. Available at: https://www.networkworld.com/article/2163430/cloud-computing/paas-primer-what-is-platform-as-a-service-and-wh□-does-it-matter-.html, 10.01.2018.

[13]. Kadam M, Jidge P., Tambe S., Ta□ade E. Cloud Service Based On Database Management S□stem , Int. Journal of Engineering Research and Applications. ISSN:2248-9622, Vol. 4, Issue 1( Version 3), Januar□2014, pp. 303-306

[14]. Oracle Infrastructure and Platform Cloud Services Securit□ Oracle white paper, November 2016. Available at: https://cloud.oracle.com/opc/iaas/whitepapers/Oracle_Cloud_Securit□_Whitepaper.pdf, 10.01.2018.







[15]. Achieving HIPAA compliance with Postgers Plus Cloud Database. EnterpriseDB white paper. EnterpriseDB Corporation, 2015. https://www.enterprisedb.com/hipaa-compliance-postgres-plus-cloud-database 10.01.2018.

[16]. Online Database Software. Custom Database Applications. Caspio. Available at: https://www.caspio.com/, 10.01.2018.

[17]. ClearDB - The Ultra Reliable, Geo Distributed Data Services Platform. Available at: http://w2.cleardb.net/ , 10.01.2018.

[18]. Sk□SQL Makes Highl□Available Databases Eas□, with MariaDB Enterprise | MariaDB Available at: https://mariadb.com/about-us/newsroom/press-releases/sk□sql-makes-highl□-available-databases-eas□-mariadb-enterprise, 10.01.2018.

[19]. Nikola□enko A. Year of cloud DBMS. Open S□stems. DBMS, ISSN 1028-7493, 2013, No 9 (in Russian), Available at: https://www.osp.ru/os/2013/09/13038286/ , 10.01.2018.

[20]. Shared disk architecture – Wikipedia. Available at: https://en.wikipedia.org/wiki/Shared_disk_architecture, 10.01.2018.

[21]. Shared-nothing architecture – Wikipedia. Available at: https://en.wikipedia.org/wiki/Shared_nothing_architecture, 10.01.2018.

[22]. Parallel Execution with Oracle Database 12c Fundamentals. Oracle White Paper, December 2014. Available at http://www.oracle.com/technetwork/database/bi-datawarehousing/twp-parallel-execution-fundamentals-133639.pdf, 10.10.2018.

[23]. Taniar D., Leung C. H. C., Raha□u W., Goel S. High Performance Parallel Database Processing and Grid Databases. Ch. 10. Wile□Publishing, 2008, isbn: 9780470107621, pp. 289-320

[24]. Bauer M. Oracle8i Parallel Server Concepts, Release 2 (8.1.6), part No. A76968-01, December 1999. Available at: https://docs.oracle.com/cd/A87860_01/doc/server.817/a76965.pdf, 10.01.2018.

[25]. Oracle Active Data Guard, Real-Time Data Protection and availabilit□ Oracle White Paper, October 2015. Available at: http://www.oracle.com/technetwork/database/availabilit□/active-data-guard-wp-12c-1896127.pdf, 10.01.2018.

[26]. Oracle GoldenGate 12c: Real-Time Access to Real-Time Information. Oracle White Paper, March 2015. Available at: http://www.oracle.com/us/products/middleware/data-integration/oracle-goldengate-realtime-access-2031152.pdf, 10.01.2018.

[27]. Chu T. Top Five Reasons to Choose SharePlex® Over Oracle GoldenGate. Quest Software, November, 2011. Available at: http://www.dlt.com/sites/default/files/Quest-Shareplex-Whitepaper.pdf, 10.01.2018.

[28]. Percona XtraDB Cluster Release5.7.17-29.20 Operations Manual Available at: https://learn.percona.com/download-percona-xtradb-cluster-5-7-manual, 10.01.2018.

[29]. Xiai Yan, Jinmin Yang, Qiang Fan. An Improved Two-phase Commit Protocol Adapted to the Distributed Real-time Transactions. Przegląd Elektrotechniczn□ (Electrical Review), ISSN 0033-2097, R. 88 NR 5b/2012, pp. 27-30

[30]. Bernstein P. A., Hadzilacos Goodman N.: Concurrenc□ Control and Recover□ in Database S□stems, Addison Wesle□ Publishing Compan□, ISBN 0-201-10715-5. 1987 pp. 49-53.

[31]. Open Group Standard DRDA, Version 5, Volume 3: Distributed Data Management (DDM) Architecture // ISBN: 1-931624-93-3 Document Number: C114. pp. 831-832

[32]. Gra□J., Lamport L. Consensus on Transaction Commit. Microsoft Research. 1 Januar□ 2004 revised 19 April 2004, 8 September 2005. Available at:

https://www.microsoft.com/en-us/research/publication/consensus-on-transaction-commit/, 10.01.2018.

[33]. Mahmoud H. A., Arora V., Nawab F., Agrawal D., El Abbadi A. Maat: Effective and scalable coordination of distributed transactions in the cloud. Proceedings of the VLDB Endowment, Volume 7, No 5. Januar□2014, pp. 329–340

[34]. Keidar I. Dolev D. Increasing the Resilience of Distributed and Replicated Database S□stems.Journal of Computer and S□stem Sciences (JCSS). December 1998, Issue 57 (3), pp. 309–324. DOI:10.1006/jcss.1998.1566

[35]. M□SQL :: M□SQL 5.7 Reference Manual :: 21 M□SQL NDB Cluster 7.5 and NDB Cluster 7.6. Available at: https://dev.m□sql.com/doc/refman/5.7/en/m□sql-cluster.html, 10.01.2018.

[36]. Das S., Agarwal S., Agrawal D., El Abbadi A. ElasTraS: An Elastic, Scalable, and Self Managing Transactional Database for the Cloud. UCSB Computer Science Technical Report 2010-04, pp. 1-14.

[37]. The Teradata Scalabilit□Stor□, A Teradata White Paper, EB-3031 0701, 2001, NCR Corporation. Available at: http://www3.cs.ston□brook.edu/~sas/courses/cse532/fall01/teradata.pdf, 10.01.2018.

[38]. Gridscale® Database Virtualization Software. Technical whitepaper, xkoto, Inc. Item: GS-WP-EN-20080930. 2008. Available at: http://www.tech-21.com.hk/download/Gridscale_Technical_White_Paper.pdf , 10.01.2018.

[39]. Michalewicz M., Clouse B., McHugh J. Oracle Real Application Clusters (RAC). Oracle White Paper. June 2013. Available at: http://www.oracle.com/technetwork/database/options/clustering/rac-wp-12c-1896129.pdf, 10.01.2018.

[40]. Kuznetsov S. D. Transactional Massive-Parallel DBMSs: A New Wave. Trud□ ISP RAN/Proc. ISP RAS, 2011, vol. 20, pp. 189-251.

[41]. Stonebraker M., Madden S., Abadi D.J., Harizopoulos S., Hachem N., Helland P. The End of an Architectural Era (It's Time for a Complete Rewrite). Proceedings of VLDB, 2007, Vienna, Austria, pp. 1150-1160.

[42]. Boichenko A.V., Rogojin D.K., Korneev D.G. Algorithm for d□namic scaling relational database in clouds. Ekonomika, statistika I informatika. [Econom□, statistics and informatics], ISSN 2500-3925 (Print), 2014. No 6 (2), pp. 461-465 (in Russian).

[43]. Chistov V.A., Luk□anchenko A.V. Automation of scaling high loaded M□SQL databases. Sovremenn□e nauko□omkie tekhnologii [Modern high-tech], ISSN 1812-7320, 2016, 6-2, pp. 315-319 (in Russian).

[44]. Gorobets V.V. Mathematical models and algorithms for optimizing the distribution of transaction s□stem data. Thesis for a Candidate of Technical Sciences degree, Platov South-Russian State Pol□technic Universit□(NPI), Novocherkassk, 2015. 210 pages (in Russian).

[45]. Zernov A.S., Ozhiganov A.A. Horizontal scaling of database using consuistent hashing. Izvesti□a v□sshih uchebn□h zavedeni□ Priborostroenie. [higher educational institutions news. Instrumentation], ISSN 2500-0381, 2017. vol. 60. № 3. pp. 234-238 (in Russian).

[46]. Caio H. Costa, João Vianne□, Paulo Maia, Francisco Carlos M. B. Oliveira. Sharding b□ Hash Partitioning - A Database Scalabilit□Pattern to Achieve Evenl□Sharded Database Clusters. 17th International Conference on Enterprise Information S□stems (ICEIS 2015), At Barcelona, Spain. DOI: 10.5220/0005376203130320





[47]. Gusev E.I. Implementation sphere researching of distributed transaction nonblocking commit algorithm. Visn□k NTUU “KPI” Informatics, operation and computer science, ISSN 0135-1729, 2012, Issue 57, pp. 76-80 (in Russian).

[48]. InfiniBand – Wikipedia. Available at: https://en.wikipedia.org/wiki/InfiniBand, 10.01.2018

[49]. Gusev E.I. Optimization of access to distributed pages in a cloud computing s□stems based on shared ever□thing architecture using unload queue method. Problem□ informat□zatsii ta upravlinnia. [Problems of informatization and management], ISSN 2073-4751, 2015. vol. 4, No 52. pp.17-21 (in Russian).