К РЕТРОСПЕКТИВНОЙ ОСНОВЕ ДОЛГОВРЕМЕННОЙ ЭФФЕКТИВНОСТИ РАСПРЕДЕЛЕННЫХ СЕРВИСОВ

С.В. Знаменский, зав. лаб., доц. Института программных систем им. А.К. Айламазяна РАН, д-р физ.-мат. наук

Анализируются тенденции развития теории информационных систем и описывается направление исследований, открывающее путь к комплексному решению базовых проблем теории в перспективе 15–20 лет.

Ключевые слова: надежность, жизнестойкость, катастрофоустойчивость, самовосстановление, устойчивое развитие, социотехнические системы.

В различных сферах человеческой деятельности растет потребность в надежных постоянно доступных информационных сервисах, обеспечивающих каждого сотрудника или потребителя качественным доступом к информации. Разработка таких сервисов обычно базируется на стандартах, основанных на исследованиях опыта предыдущих разработок и теоретически обеспечивающих требуемые характеристики качества и надежности.

Провал, не покрытый стандартами. На практике долговременные потери доступа остаются привычно случающимися даже в уникальных системах повышенной надежности, гарантированно устойчивых к единичным поломкам оборудования. Примером является рынок ценных бумаг США, объединяющий множество компьютерных агентов частной торговли (High Frequency Trading). Финансовый кризис 6 мая 2010 г., названный «Восстанием компьютеров» (Rise of the machines) и «молниеносным крушением» (Flash Crash), ярко продемонстрировал, что качество стандартов и их соблюдение не предохраняют от побочных эффектов взаимодействия компьютерных систем, создающих ситуацию, не имеющую аналогов в прошлом [1].

Прогресс (как экономический, так и технический) невозможен без сосуществования и тесного взаимодействия автономно развивающихся социотехнических систем. Надежной верификации регламента этого взаимодействия на содержательном уровне препятствует неизбежная неполнота понимания сложной изменчивой природы внешней среды.

Вывод о необходимости поиска новых подходов к разработке программного обеспечения, позволяющих на практике обеспечить безупречное непрерывное обновление сложных систем (и, значит, корректное сосуществование разных версий подсистем), регулярно делается в отчетах ведущих исследовательских центров [2] и научных публикациях [3, 4]. В [5] приведено исследование динамики развития научных концепций в области информационных систем, результаты которого однозначно показывают сдвиг интереса правительств западных стран в последние два года к обострившимся проблемам сюрпризов, неожиданных побочных эффектов и результатов новых технологий.

Проблема гладкой обновляемости взаимодействующих систем. Сегодняшние технологии и методологии разработки программного обеспечения ориентированы максимум на дальнейшую доработку, но ни в коем случае не на частичную переделку. Поэтому необходимость в замене возникает каждые 2–5 лет. Это хотя и кажется выгодным фирмам – производителям информационных систем, но создает головную боль для клиентов, уже страдавших от переходных процессов и ностальгии по привычным интерфейсам и другим достоинствам предыдущих систем.

Столь же неотвратимым источником непредвиденных потерь доступности является обновление устаревшей системы, тормозящее и частично блокирующее работу пользователя. Реальные потери при обновлениях информационных систем являются коммерческой тайной. Конфигурирование систем повышает адаптивность и снижает остроту проблемы в крат-

ковременной перспективе. Однако это безупречно срабатывает лишь при условии, что структуры данных старой и новых систем достаточно близки. Постепенно нарастает необходимость столь существенных структурных изменений, что резкого падения надежности в фазе обновления избежать практически невозможно. Любая нештатная ситуация, наложившись в этой фазе на ошибку организации обновления сложной уникальной системы, способна привести к нежелательным последствиям вплоть до крупномасштабной техногенной катастрофы.

Ситуация драматически ухудшается, когда системы начинают взаимодействовать, в частности, управление конфигурированием осуществляет автоматическая независимо разработанная подсистема [6]. Если учесть рост количества независимо разрабатываемых программ и разнообразия их взаимосвязей, то становится очевидным, что будущее за сложными системами, подсистемы которых разрабатываются независимо и обновляются автономно. Уже сейчас каждое обновление чревато неожиданными побочными эффектами, а с усложнением взаимосвязей сюрпризы участятся. Поэтому возможности продуктивной реализации сложных систем в значительной степени определятся принципиальной основой организации обмена данными.

Отсюда новая цель исследований: создание основы долговременно надежного эффективного распределенного сервиса с регулярно локализованно улучшаемой логикой.

Локализованное улучшение логики – это уровень сложности системы, успешно адаптируемой для совместной работы группами людей, не полностью понимающими друг друга. Разные субъекты (индивидуалы и организации) смогут в такой системе реализовывать свою активность правильным в их понимании способом и при осознании необходимости самостоятельно (с участием программистов) перестраивать структуру своей активности. Такие перестройки в рамках совместной договоренности практически не будут заметны остальным субъектам (кроме непосредственно вовлеченных в реорганизацию). Подобно привычному представлению функции сплайн, эклектичная логика поведения системы представляется иерархией независимых логик поведения субъектов системы.

Два тупика на выбор системного архитектора. Слепая вера в символичность так называемой «теоремы CAP» (подробнее о которой см. [7]) фактически разделила системы обработки данных на два класса.

Сторонники безупречной логики реляционных баз данных видят развитие в исключении задержек обработки [8] с концентрацией вычислений в одной установке. Развитие этого базового направления связано с использованием новых технологий устройств хранения информации, организации упреждающих (tentative) вычислений [9] и асинхронной обработки коммутирующих изменений [10].

Никто не сомневается в универсальности логического подхода. Однако головокружительный успех таких фирм, как Yahoo, Amazon, Google, Yandex, жертвующих строгостью логики в пользу ускорения отклика системы, стимулирует интенсивные исследования в альтернативном направлении, нацеленном на ускоренную обработку больших объемов информации и выдачи «в конечном итоге согласованного» (Eventually consistent) ответа.

Сторонники альтернативного подхода (BASE) отмечают принципиальную ограниченность систем, основанных на транзакциях, распространяющуюся не только на последовательное исполнение (принципы ACID), но и на вышеупомянутое ускорение асинхронной обработкой коммутирующих изменений (принципы CALM). В [11] справедливо замечено, что эти принципы неприменимы при наличии:

– длительных транзакций, обработка которых может продолжаться днями или неделями;

– автономных участников независимых систем (подсистем), которые могут не оправдать ожидания или потерять информацию о состоянии;

– трудно искоренимой возможности непредусмотренного дублирования реплик, приводящего к опасному повторному выполнению операции.

Качество систем, основанных на BASE, неуклонно растет. Но кратковременные рассогласования при взаимодействии систем время от времени будут неизбежно приводить к непредусмотренным побочным эффектам.

С точки зрения описанной выше проблемы гладкой обновляемости взаимодействующих систем, категорически неприемлемы оба подхода: BASE – из-за непредсказуемости последствий кратковременного рассогласования, а ACID и CALM – из-за нетерпимости к задержкам, неизбежным при выполнении.

Любая их комбинация наследует все недостатки каждого. Нужна новая идея.

Идея и предрассудки. Высокая доступность истории позволила Википедии резко опустить традиционно высокую планку требований к источникам информации и позволить анонимным пользователям редактировать тексты. Она обеспечила высокую исправляемость ошибок и защищенность от вандализма, что сделало Википедию крупнейшей в мире энциклопедией, непревзойденной по объему и полноте представленной информации, средствам навигации, темпам качественного роста.

Почему бы не снизить требования к квалификации программистов, участвующих в разработках прикладных сервисов, тем же путем, которыми Википедия снизила требования к авторам публикаций без ущерба для результата? В расширившийся круг участников разработки естественно включатся старшеклассники и младшекурсники, проявившие интерес к участию в серьезных прикладных разработках. Это обеспечит им реальные дела с реальными постановками задач от реальных пользователей, тщательной проверкой результатов, критикой и одобрениями, обменом опытом и другими атрибутами качественной программистской практики, на которой единственно только и может основываться обучение программиста.

Дело за системой, защищающей исполнение программного кода подобно тому, как в Википедии защищается текст. Разумеется, написание и отладка хорошего программного кода – это процесс с гораздо более тесными связями и взаимозависимостями, требующий намного более основательной поддержки, чем создание текстов Википедии. Он, в отличие от процесса создания энциклопедии, не осуществим в современной парадигме разработки информационных систем, представляющей систему набором изолированных компонент, обменивающихся системными сообщениями. В возможность его реализации сейчас мало кто верит, как десять лет назад мало кто верил в успех Википедии. В России с подачи лидера российской суперкомпьютерной отрасли чл.-корр. РАН С.М. Абрамова [12] и вице-президента Российской академии образования В.А. Болотова [13] эта задача прорабатывалась в цикле исследований преподавателей и студентов университета г. Переславля.

Проработка выделила ряд устаревших азбучных истин тех времен, когда распределенных компьютерных систем не существовало:

1. Большая система делится на изолированные компоненты, последовательно обменивающиеся сообщениями.
2. Исполняемый код должен быть отлажен до запуска в системе.
3. Свойства согласованности реплик, доступности системы и устойчивости к разделению (CAP) совместить в одной системе невозможно.
4. Непосредственный доступ к неискаженной истории изменений в распределенной системе за длительное время потребует несоразмерных ресурсов, а полная история изменений слишком велика, чтобы ее можно было сохранять в доступном виде.

Первая истина отражает представление о логической целостности системы, унаследованное от однопроцессорных систем, и за отсутствием успешных альтернатив считается окончательно доказанной мировой практикой разработки ПО. Моделирование сложной системы набором искусственно изолированных модулей приводит к неадекватности модели, выражающейся в раздражающей пользователей бестолковости системы, устранение которой тре-

бует коррекции общей архитектуры, опирающейся на понимание сложности всех взаимосвязей одним человеком, что для реальных задач часто выходит за рамки человеческих возможностей. Желание обеспечить гладкость обновления увеличивает сложность и усугубляет ситуацию.

Вторая истина также осталась от неэволюционирующих систем. Разработка направленно эволюционирующей системы может быть продуктивной только, если она ведется в самой системе, сокращая дистанцию между пользователем и группой сопровождения, обеспечивая оперативность исправления мелких ошибок, учет затрат и эффективности труда программистов и ориентируя разработчика на значимый для пользователей результат.

Популярная трактовка упомянутой теоремы CAP гласит, что система не может одновременно сочетать свойства доступности, безупречной логичности и устойчивости к временным разделениям системы (потерям связи). На самом деле в такой обобщенной форме она не верна. В [14] приведен контрпример и описана система, обладающая всеми тремя свойствами. Полное сохранение логики обработки эта система сочетает с устойчивостью к задержкам связи, что позволяет в сто раз ускорить обработку запросов по сравнению с аналогичной системой, построенной на принципах ACID, в полном объеме предоставляя ее функциональность.

Последний шаблон также имеет в основании четко сформулированные и безупречно доказанные теоремы теории алгоритмов. Однако с точки зрения формальной логики для двух последних имеет место логическая ошибка, называемая подменой тезиса: доказанные утверждения существенно отличаются от тех популярных интерпретаций, на которых базируется их применение. Сохранение неискаженной истории изменений практически реализовано в СУБД Oracle, а пользовательский доступ к такой истории – «машина времени» в операционной системе фирмы APPLE.

Новое видение теории информационных систем предполагает замену устаревших предрассудков новым пониманием модульности, согласованности и структур данных.

Модульность социотехнических систем. Большие социотехнические системы предельно сложны для понимания. Один субъект (программист или группа программистов) не способен безупречно регламентировать эффективную эволюцию взаимодействия между составляющими. Поэтому декомпозиция системы должна не только обеспечить видимые потребности, но и предоставить широкие возможности неограниченной адаптации к еще невыявленным требованиям на любом уровне организации. Архитектура системы должна обеспечивать изменение структур данных и алгоритмов их обработки без побочных эффектов для не связанных напрямую процессов.

Децентрализация разработки редуцирует сложность управления эволюцией информационной системы до непрерывного рефакторинга и реинжиниринга одного процесса активности социотехнической системы.

Системная поддержка децентрализации разработки открывает неизведанный путь к созданию и поддержке систем, сложность которых ограничивается лишь количеством вовлеченных субъектов разработки.

Для этого декомпозиция информационной системы на модули должна так отражать разделение труда и внутреннюю организацию социотехнической системы, чтобы каждая активность, связанная с самостоятельной территорией, организацией, подразделением, рабочей группой или направлением работ в социотехнической системе, поддерживалась автономным модулем информационной системы со своими алгоритмами и структурами данных. Одной из таких активностей становится сопровождение информационной системы, включающее дальнейшую доработку.

Автономность здесь означает, что алгоритмы и структуры данных могут в любой момент быть изменены без ущерба для не связанных напрямую активностей. Далеко не все процессы-активности и соответственно модули будут автономными. Поведение большинства из них

детерминировано родительской (руководящей) активностью и самостоятельной разработкой не занимается. Активность может по необходимости приобретать или терять статус автономности.

Существенная перестройка модуля реализуется в сотрудничестве руководства соответствующего подразделения с субъектом разработки информационной системы. Субъект разработки при этом:

– изучает и контролирует лишь часть системы, качественно обслуживая конкретную задачу иерархии задач системы, не касаясь остальных задач;

– описывает и соблюдает формат и содержание выходных данных своей задачи, имея возможность их изменить в новой версии, помня о том, что каждую версию ему придется приоритетно сопровождать до тех пор, пока от нее не откажутся все потребители.

Информировать о целях эволюции и изменениях в структуре новой версии сервиса надо только подписчиков этого сервиса. Стандарты и схемы, описывающие рамки взаимодействия, в любой момент могут по соглашению сторон быть заменены на обновленные версии. Проблема конкурентности доступа сменяется проблемой оценки качества и обновлений списка источников информации.

В штатном режиме работы поддержка активности должна осуществляться на узлах, географически близких к субъектам активности, но реплицированные резервные сервисы обычно размещаются в значительном удалении. Передислокация сервиса может не требовать изменения его программного кода и не угрожать потерей даже доли последних данных. Контекстно автономная система может начать функционирование на одном узле и постепенно распространиться по континентам, а в отдаленном будущем сможет спрятаться в музейный архивный ЦОД.

Эти идеи в разной степени применяются в разных системах. *Принцип контекстной автономности* означает их комплексную поддержку в архитектуре системы на низком уровне, обеспечивающую ([15]):

– высокую согласованность изменяемых данных (это особенно важно для версий кода);

– защищенность истории изменений данных и возможность расследования происшествий на низком уровне (программисты не могут изменить историю, но могут использовать прежние версии);

– безопасную отладку сырого кода в работающей системе.

Ретроспективная основа согласованности данных. Суть ретроспективной парадигмы в мужестве отказа от понятия «текущее состояние распределенной системы». Не секрет, что изменения данных в распределенной системе происходят не мгновенно, и состояние в момент обработки *не определено*. На то, чтобы выяснить наличие в данный момент времени согласованного состояния у большой системы, требуется время. Поэтому моделирование работы сложной системы привычными методами теории конечных автоматов *не корректно*.

Лишь в определенные периоды времени система имеет корректно согласованное состояние. В другие моменты этого состояния нет (оно принципиально недоступно подобно тому, как в квантовой механике принципиально недоступны точные значения координат и импульса частицы). Ответ на запрос о состоянии данных, сделанный в неудачный момент, мы можем получить либо, дождавшись согласованного состояния, либо на основе сохраненной информации о прошлом согласованном состоянии. Разумность ожидания сомнительна уже потому, что пока ответ дойдет, пройдет некоторое время, за которое состояние иногда может измениться.

Обычно возражают, что все эти промежутки времени настолько малы, что ими можно пренебречь. Но даже если это так в штатном режиме, то любая задержка в нештатной ситуации может повлечь неожиданное поведение хорошо отлаженной системы в самый критический момент.

Итак, у системы (и ее подсистем) состояние определено не всегда. Это может стать проблемой при сложении значения A из одной подсистемы со значением B из другой. Например, если A и B доступны поочередно каждый раз с новым значением переменных, то ответ вряд ли будет правильным. Верный ответ получится, только если оба значения доступны одновременно.

Выход остается один: сохранять информацию о доступных состояниях и их времени и использовать последний момент времени, на который все операнды были доступны, а результат операции помечать тем же временем, поскольку он относится к тому же состоянию системы.

Ретроспективную основу, таким образом, составляют:

1. Единый набор вложенных шкал с разным шагом, например, от начала 1970 г. с шагами в 2^k с.

2. Выбор шкалы для штатного режима; одного шага должно быть достаточно для обработки.

3. Синхронизация системного времени точнее полшага используемой шкалы.

4. Подсистема низкого уровня, эффективно хранящая и возвращающая предыдущую консистентную информацию по запросу на любой момент прошлого и сигнализирующая о наличии необработанных изменений.

Техническую проблему представляет именно последний пункт, требующий самостоятельной проработки. Подход к его реализации описан в [16], но он требует технической проработки и экспериментальных исследований.

Разумеется, ныне разрабатываемые системы в массе своей не нуждаются в ретроспективной основе. Но только такая основа сделает возможной разработку сложных и гибких, быстрых и высоконадежных систем мониторинга экономики, окружающей среды и безопасности жизнедеятельности для будущего усложняющегося мира.

Список литературы

1. **Sommerville I., Cli D., Calinescu R. et al.** Large-scale Complex IT Systems // Communications of the ACM, 2012, V. 55, № 7.

2. **Feiler P., Gabriel R.P., Goodenough J. et al.** Ultra-Large-Scale Systems: The Software Challenge of the Future. Technical Report. Carnegie Mellon University Software Engineering Institute, 2006.

3. **Ncube C.** On the Engineering of Systems of Systems: key challenges for the requirements engineering community // Requirements Engineering for Systems, Services and Systems-of-Systems, 2011, August.

4. **Strigini L.** Fault tolerance and resilience: meanings, measures and assessment // Resilience Assessment and Evaluation of Computing Systems, eds. K. Wolter et al., London: Springer, 2012.

5. **Merali Y., Papadopoulos T., Nadkarni T.** Information systems strategy: Past, present, future? // J. Strateg. Inform. Syst., 2012, June, V. 21, № 2.

6. **Ghafari M., Jamshidi P., Shahbazi S. et al.** An architectural approach to ensure globally consistent dynamic reconfiguration of component-based systems // Proceedings of the 15th International ACM SIGSOFT Symposium on Component-based Software Engineering (CBSE'2012). Bertinoro, Italy, June 2012.

7. **Кузнецов С.Д.** Транзакционные параллельные СУБД: новая волна. URL: http://citforum.ru/database/articles/kuz_oltp_2010/.

8. **Nawab F., Agrawal D., El Abbadi A**. Message Futures: Fast Commitment of Transactions in Multi-datacenter Environments // 6th Biennial Conference on Innovative Data Systems Research (CIDR'13). Asilomar, California, USA, January 6–9, 2013.

9. **Muehe H., Kemper A., Neumann T.** Executing Long-Running Transactions in Synchronization-Free Main Memory Database Systems // 6th Biennial Conference on Innovative Data Systems Research (CIDR'13). Asilomar, California, USA, January 6–9, 2013.

10. **Conway N., Marczak W.R., Alvaro P. et al.** Logic and Lattices for Distributed Programming. Storming Media LLC. Technical rept. A309365, 22 June 2012, 18 p.

11. **Helland P., Haderle D.** Engagements: Building Eventually ACiD Business Transactions // 6th Biennial Conference on Innovative Data Systems Research (CIDR'13). Asilomar, California, USA, January 6–9, 2013.

12. **Абрамов С.М., Живчикова Н.С., Знаменский С.В. и др.** Архитектура системы для разработки технологий организации сложной совместной деятельности // Прикладная информатика, 2010, № 2(26).

13. **Болотов В.А., Знаменский С.В.** Требования к информационной системе управления качеством образования // Программные системы: Теория и приложения: электрон. научн. журн., 2010, Т. 1, № 2(2).

14. **Знаменский С.В.** На пути к новой теории информационных систем // Электронные библиотеки: перспективные методы и технологии, электронные коллекции (RCDL-2012). Переславль-Залесский, 2012.

15. **Знаменский С.В.** Ретроспективная основа совместной реорганизации сложных информационных ресурсов // Электронные библиотеки: перспективные методы и технологии, электронные коллекции (RCDL-2011). Воронеж, Воронежский государственный ниверситет, 2011.

16. **Знаменский С.В.** Ретроспективная основа распределенной памяти для изменчивой вычислительной среды // Материалы VI Международной конф. «Параллельные вычисления и задачи управления» (PACO'2012). Т. 2, М.: ИПУ РАН, 2012.