УДК 62-004.75

ЖИЗНЕННЫЙ ЦИКЛ СОВРЕМЕННЫХ КОРПОРАТИВНЫХ ПРИЛОЖЕНИЙ И УПРАВЛЕНИЕ НА ОСНОВЕ ПОТОКА СОБЫТИЙ

О.В. Логиновский
Южно-Уральский государственный университет (национальный исследовательский университет)
Россия, 454080, Челябинск, Ленина пр., 76
E-mail: loginovskiyo@mail.ru

А.А. Шинкарев
Южно-Уральский государственный университет (национальный исследовательский университет)
Россия, 454080, Челябинск, Ленина пр., 76
E-mail: sania.kill@mail.ru

Ключевые слова: корпоративные приложения, управление, жизненный цикл, надежность, масштабируемость, поддерживаемость, поток событий.

Аннотация: В современных условиях все большее значение обретают данные и способность работать с ними эффективно. Это касается больших данных и умения извлечь из них полезные сведения, правильно задать вопросы. Информация, которую можно извлечь из данных, необходима бизнесу, поскольку обеспечивает его выживаемость в конкурентной среде. Однако, лишь умения извлечь нужную информацию недостаточно, необходимо также выстраивать информационную инфраструктуру, способную расширяться, обновляться и снабжать данными части системы, ответственные за поддержку принятия решений и анализ.

1. Введение

Жизненный цикл современных корпоративных приложений представляет собой итеративный процесс, включающий в себя сбор требований, разработку, тестирование и поддержание работы существующей функциональности. Эти этапы повторяются вновь и вновь в течение всей жизни информационной системы.

Сегодня же, помимо описанных этапов, добавляется шаг анализа данных, артефактов работы приложения. Под артефактами подразумевается разнообразная информация о действиях пользователей, особенностях их поведения, состоянии системы в различные моменты времени, логи ошибок (error logs) и т.д. Во многих случаях данных разного рода накапливается много, и они переходят в разряд больших данных, которые позволяют ответить на еще несуществующие вопросы, выявить скрытые закономерности и в конечном итоге принести выгоду бизнесу.

Современные примеры систем поддержки принятия решений, таргетирования рекламы, прогнозирования необходимых размеров товарных запасов, начиная от ресторанов и заканчивая промышленными предприятиями, показывают, насколько новый вид «умного» анализа может быть распространен на разные сферы жизнедеятельности. Безусловно, говоря о больших данных как таковых, имеет смысл рассматривать их в

связке с методами и моделями машинного обучения, математической статистки, теорией вероятностей и визуализацией данных. Эти инструменты необходимы для того, чтобы анализ массивов данных и извлечение из них важной информации были возможны.

Можно сколь угодно говорить о впечатляющих достижениях в этой области, которые есть уже сейчас, однако, для руководителей крупных организаций, для лиц, принимающих решения и несущих за них ответственность, важна достоверность, возможность заглянуть в прошлое, спрогнозировать динамику в будущем. Достижение этих целей невозможно без хранения всей истории изменений состояния системы, а не только ее последнего состояния, как это в основном реализовано сейчас, а это, в свою очередь, подразумевает наличие точек расширения (extension points), масштабируемость (scalability), поддерживаемость (maintainability) и надежность (reliability) разрабатываемых решений.

2. Основные характеристики корпоративных приложений

2.1. Надежность

Понятие надежности агрегирует в себе такие характеристики, как способность восстановиться после сбоев, устойчивость к хакерским атакам и непротиворечивость состояния при пользовательских, программных и аппаратных ошибках.

Таким образом, система надежна, если работает правильно. Нет смысла создавать приложение, которое будет способно пережить уничтожение всех серверов, на которых развернуто. Однако, необходимо стремиться обрабатывать известные виды сбоев на ранних этапах разработки системы. Осмыслить и переписать большую систему сложнее, чем заложить в фундамент долю здоровой паранойи на старте работ.

С точки зрения сбоев аппаратного обеспечения, надежность тесно связана с масштабируемостью, которая обсуждается далее. Масштабирование хранилища данных, помимо повышения пропускной способности, также защищает от потери данных из-за сбоя жестких дисков. Не все аппаратные сбои связаны с проблемой выхода из строя хранилища данных, но именно потеря информации несет серьезные репутационные и финансовые риски [1].

Программные ошибки можно найти, всего лишь изучив кодовую базу приложения, но дьявол кроется в деталях. Факторы размера кода приложения, квалификации и усидчивости тех, кто этот код проверяет, интеграция со сторонними сервисами и проблема гонок в автоматах (race conditions) сводят теоретическую возможность выявить все ошибки еще до запуска кода к нулю. Часто программные ошибки в течение долгого времени могут оставаться незамеченными, потому что требуется определенное стечение обстоятельств, условия их срабатывания.

Таким образом, для обеспечения надежности разрабатываемой системы как минимум необходимо тестирование логики работы приложения и самого исходного кода, а также резервирование данных и хранение полного лога их изменений. Оба этих требования следует выполнять, начиная с ранних сроков разработки, и встраивать их в общий процесс на постоянной основе.

2.2. Масштабируемость

Выделяют два вида масштабируемости информационных систем: вертикальную (vertical) и горизонтальную (horizontal или shared nothing) [2].

При вертикальной масштабируемости возможное повышение пропускной способности (throughput) и производительности достигается за счет повышения мощности от-

дельно взятого сервера, например, за счет увеличения объема оперативной памяти или замены процессора на более мощный.

Горизонтальная же масштабируемость предполагает, что при добавлении нового сервера нагрузка, с которой может справляться система, возрастает. Вариант английского названия shared nothing хорошо отражает, что серверы не разделяют между собой общие ресурсы, такие как время центрального процессора, оперативную память или жесткие диски, то есть независимы друг от друга.

С одной стороны, давно считается, что вариант повышения производительности одного-единственного сервера, вместо использования нескольких относительно маломощных станций, изжил себя, потому как частота отдельного ядра процессора уже не повышается в значительной степени, как раньше [3].

С другой стороны, сейчас можно наблюдать новые линейки процессоров для настольных компьютеров, которые имеют по 16-18 ядер и соответственно 32-36 потоков выполнения при средней частоте одного ядра равной 3 ГГц, что значительно превосходит «классические» четырехъядерные процессоры с частотой 3,2-3,6 ГГц. Таким образом, можно заметить, что даже классическое вертикальное масштабирование идет по пути горизонтального масштабирования процессора, за счет увеличения количества ядер, нежели увеличения частоты их работы, которая имеет физическое ограничение при текущей архитектуре [4].

Так или иначе, рассматривать вариант вертикального масштабирования в текущих реалиях возрастающих требований к отказоустойчивости, пропускной способности, возможности осуществлять обновления без прекращения обслуживания (rolling upgrades) становится техническим долгом с самого начала. Что и говорить, если еще Мартин Фаулер в 2002 году обсуждал в своей книге архитектуру горизонтально масштабируемых систем, как наиболее предпочтительную [5].

Таким образом, разработку монолитных приложений, которые изначально не предполагают возможность своего масштабирования, уже можно смело считать антипаттерном (anti-pattern). Сервисная же архитектура, обладающая независимостью развертывания (deployment) блоков и их горизонтальной масштабируемостью, становится общепринятым подходом при разработке сложных информационных систем. Стоит отметить, что не нужно выстраивать сервисную архитектуру в простых приложениях, где это принесет больше сложностей, чем пользы от такой «гибкости».

2.3. Поддерживаемость

Помимо критериев надежности и масштабируемости, то, насколько гибка архитектура системы, насколько легко адаптируется к изменяющимся функциональным и техническим требованиям со стороны бизнеса, определяет, сможет ли бизнес быстро перестраиваться и оставаться конкурентоспособным там, где жизненный цикл проектов измеряется годами и даже десятилетиями.

Модульная система сервисов или же микросервисный подход [6] обладает несколькими преимуществами, такими как независимое развертывание, слабая связанность, возможность реализовывать решение каждой задачи, используя подходящие инструменты, языки программирования, способы взаимодействия и базы данных [7].

При использовании методологии разработки программного обеспечения DDD (Domain Driven Design) и делении приложения на ограниченные контексты (Bounded Context), возможен переход к одной базе данных на один сервис [8].

На поддерживаемость приложения влияют различные факторы: это и используемые технологии, и культура написания кода, и покрытие тестами, и архитектурные решения, принятые на ранних этапах жизненного цикла проекта. Общего рецепта, как сделать систему поддерживаемой, расширяемой и управляемой, пожалуй, нет. Но

управлять сложностью проекта, его техническим долгом, адекватно оценивать будущие точки расширения, проводить непрерывное тестирование абсолютно необходимо для того, чтобы иметь шансы не переписывать код раз в два года.

3. Потоковая обработка данных

Применяемому к большим данным машинному обучению в целом и его моделям в частности необходимы исходные «сырые» данные (raw data), и чем обширнее и качественнее они, тем результативнее работа. Поэтому важен исходный материал, которого может и не быть при привычном подходе к хранению только последнего состояния имеющихся бизнес-сущностей (business entities).

Переход от хранения только текущего состояния системы к хранению потока событий кажется весьма логичным шагом развития технологии и мировоззрения сообщества программистов, архитекторов и бизнеса в широком смысле этого слова. Ведь первоисточник в виде событий изменения состояния системы позволяет восстановить данные по состоянию на любой момент времени. Эта история изменений дает возможность строить отчеты любой сложности и производить новый анализ данных и через несколько лет после их генерации.

Однако, помимо перехода к системам, хранящим свое состояние в виде потока событий, также необходим переход от единственного реляционного хранилища данных или же имеющего синхронную репликацию, к которому сообщество привыкло и даже обрело пагубную зависимость от такой модели хранения данных и взаимодействия с ними, к распределенному хранилищу с репликацией (replication), партиционированием (partitioning, sharding), асинхронному распространению изменений, отказу от распределенных транзакций (distributed transactions), как средству, обеспечивающему определенные гарантии целостности операций, но имеющему свою цену в плане общей производительности системы и ее пропускной способности на чтение и запись.

Этот сдвиг массовой парадигмы в разработке приложений кажется таким же необходимым и неизбежным, как когда-то переход от однопроцессорных систем с одним потоком и отсутствием состояния гонок в автоматах [9] к многопоточной модели, которая накладывает определенные ограничения, требует большей внимательности и аккуратности, а также использования средств синхронизации потоков и процессов выполнения.

Безусловно, идею хранения истории изменения данных и получения унаследованного представления из них нельзя отнести к новым. Аналогичный подход имеет место, например, в реляционных базах, где хранится лог транзакций (transaction log) и WAL (write ahead log) для индексов, которые позволяют восстанавливать состояние базы данных после сбоев, взаимных блокировок (deadlocks), проводить аудит, но имеют ограниченный размер и зачастую очищаются после прохождения определенного промежутка времени, то есть не хранятся «вечно».

Идея же хранения всего потока изменений и событий, в частности Event Sourcing, не предполагает удаление старых событий для экономии дискового пространства. Напротив, случившиеся события считаются неизменяемыми (immutable), и это дает ряд преимуществ. Среди таких преимуществ можно выделить возможность глубокого анализа истории событий в системе, построение аналитики любой сложности, за счет наличия полной истории, а не только последнего актуального состояния системы, возможность кеширования событий и т.д.

4. Заключение

Какие бы направлениях развития разработки корпоративных информационных систем не рассматривались, будь то разнообразие решений для хранения данных, развитие процессоров, внедрение новых программных архитектур, везде основным трендом является уход от вертикальной масштабируемости и переход к горизонтальной. Масштабируемость вычислений и хранилищ данных, с ростом объемов данных и сложности задач по их анализу, становится краеугольным камнем дальнейшего развития разработки программного обеспечения.

Инертность внедрения новых подходов к разработке объясняет сложность сдвига мышления в сторону потоков событий, как основного варианта хранения данных, хотя и этот вариант имеет под собой хорошо известные примеры из прошлого. В частности, бухгалтерская книга, в которой записи лишь добавляются, но не исправляются после внесения, а ошибки в предыдущих записях лишь компенсируются новыми строками.

Таким образом, внедрение потока событий, в качестве хранилища данных, является необходимым условием для построения гибкой системы управления, не ограниченной только сегодняшними потребностями, рассчитанной на дальнейшее развитие.

Из-за сложностей идеологического, методологического и технического характера в подавляющем большинстве компаний построение корпоративных систем происходит с использованием подходов, которые апробировались последнее десятилетие.

Можно сказать, что существует целый ряд стандартных задач, таких как документооборот, аутентификация, авторизация, CRUD-операции для различных бизнес-сущностей (Create, Read, Update, Delete), построение отчетов и аналитики, и т.д. И решение этих задач тоже уже во многом устоялось, хоть и существует множество вариаций не только с точки зрения технологического стека, но и качества исполнения конечной реализации.

Можно однозначно сказать, что Big Data, Data Science и развитие направлений масштабируемости и отказоустойчивости информационных систем предопределили развитие информационных технологий в долгосрочной перспективе. Компании, которые не воспользуются всем спектром предоставляемых возможностей, могут остаться на плаву, но максимальную выгоду и лидирующие позиции обеспечат себе лишь те, кто инвестирует ресурсы и сможет перестроиться под стремительно меняющиеся реалии.

Список литературы

1. https://www.csoonline.com/article/3019283/data-breach/does-a-data-breach-really-affect-your-firm-s-reputation.html.
2. Liu C.Y., Shie M.R., Lee Y.F., Lin Y.C., Lai K.C. Vertical/Horizontal Resource Scaling Mechanism for Federated Clouds // Proceedings of International Conference on Information Science & Applications (ICISA). Seoul, South Korea, 2014. IEEE, 2014. P. 1-4.
3. https://habr.com/company/intel/blog/194836.
4. Gepner P., Kowalik M.F. Multi-Core Processors: New Way to Achieve High System Performance // Proceedings of Fifth International Conference on Parallel Computing in Electrical Engineering (PARELEC 2006). Bialystok, Poland, 2006. IEEE, 2006. P. 9-13.
5. Фаулер М. Рефакторинг: улучшение существующего кода. СПб.: Символ-Плюс, 2003. 432 с.
6. Namiot D., Sneps-Sneppe M. On Micro-services Architecture // International Journal of Open Information Technologies. 2014. Vol. 2, No. 9. P. 24-27.
7. https://www.youtube.com/watch?v=xJMbkZvuVO0.
8. https://martinfowler.com/articles/microservices.html.
9. https://studopedia.org/8-149146.html.