ОБЕСПЕЧЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ И ОТКАЗОУСТОЙЧИВОСТИ В СЕТЯХ МОБИЛЬНОЙ СВЯЗИ С ПОМОЩЬЮ ERLANG/OTP

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Ветер Е. В.

Ярмолик В. Н. – д-р. техн. наук, профессор

Одним из ключевых требований к программному обеспечению, функционирующему в современных сетях мобильной связи, является обеспечение высоких уровней производительности и отказоустойчивости. Данный доклад обобщает 10-летний коммерческий опыт автора в реализации высококонкурентных систем маршрутизации сообщений для сетей мобильной связи, и иллюстрирует преимущества и трудности использования платформы Erlang/OTP для решения этого класса задач.

Современная индустрия телекоммуникаций предъявляет жесточайшие требования к отказоустойчивости программного обеспечения: стандартным требованием является «99.999% доступности», что означает, что разрабатываемое ПО не может простаивать в результате отказов более 5.25 минуты в год. В то же время, стремительный рост числа пользователей приводит к многократному увеличению объёмов обрабатываемого трафика: например, количество отправленных пользователями мобильных сетей SMS-сообщений увеличилось с 7.8 трлн. в 2011 г. до 8.6 трлн. в 2012 г. [1]

Традиционный подход «горизонтального масштабирования» ограниченно применим в таких условиях – ибо кратное увеличение количества оборудования увеличивает также и вероятность аппаратных сбоев, снижая надёжность. В этой связи приходится изыскивать технологии разработки, максимально использующие возможности имеющегося оборудования.

В качестве примера рассмотрим широко распространённую в индустрии задачу маршрутизации СМС-трафика корпоративных клиентов для среднестатистического оператора сотовой связи с несколькими миллионами абонентов. Как показывает практика, для достаточного покрытия потребностей такой абонентской базы в часы пиковой нагрузки, система маршрутизации должна обрабатывать не менее тысячи сообщений в секунду. Обработка каждого сообщения включает в себя следующую последовательность действий:

- Получение сообщения от клиента;
- Проведение биллинговой транзакции по счёту клиента;
- Запись в БД полной информации о сообщении для статистических отчётов;
- Проверка ограничений («чёрные списки», ограничения трафика клиента);
- Расчёт маршрута к получателю с учётом MNP (переносимости абонентского номера);
- Доставка сообщения на нужный uplink SMSC (центр обработки сообщений);
- Обновление промежуточного статуса доставки в БД;
- Ожидание отчёта о доставке от SMSC;
- Обновление окончательного статуса доставки в БД.

Все перечисленные процессы должны выполняться с очень высокой степенью конкурентности: сообщения отправляются и принимаются одновременно большим количеством клиентов мобильной сети. Аппаратный или программный сбой любого компонента системы не должен приводить к утрате функциональности.

Разработка архитектуры ПО для реализации такой системы требует решения нескольких непростых задач. Мы оставим задачи организации хранения данных в БД и доступа к ним за рамками данного доклада, и сосредоточимся на решении задач организации параллельной обработки и отказоустойчивости.

В рамках работы над проектом были разработаны и протестированы две различных архитектуры: традиционная многопоточная, реализованная средствами языка Java, а также архитектура, основанная на легковесных «виртуальных процессах», реализованная на языке Erlang.

В «традиционной» архитектуре параллельная обработка была реализована при помощи трёх видов потоков: сравнительно простые и короткоживущие потоки для обслуживания входящих (downlinks) и исходящих (uplinks) подключений, а также более сложные долгоживущие потоки для обработки биллинга, статистики и т.п.

Отказоустойчивость реализовывалась при помощи дополнительной логики в приложении: специальный управляющий процесс занимается распределением потоков между серверами. В процессе реализации пришлось столкнуться со следующими основными трудностями:

- Проблема обеспечения надёжности: сбой одного потока может повлечь за собой сбой всего приложения;
- Недостаток производительности: создание и удаление потоков на уровне ОС является довольно дорогой операцией. Эта проблема отчасти решается организацией пула готовых к выполнению потоков – однако управление пулом вносит свои трудности в разработку [2];
- Ограничения ОС: теоретически, в ОС Linux можно создать более 190 тыс. потоков, чего вполне достаточно для нагруженной системы с множеством одновременно работающих клиентов. На практике, однако, система приходит в состояние «клинической смерти» уже при

15 тыс. выполняющихся одновременно потоков;

- Стоимость разработки: синхронизация потоков достаточно нетривиальна в отладке и приводит к увеличению времени, требующегося на тестирование и разработку, а следовательно – к увеличению стоимости;
- Необходимость реализации собственной логики для распределения потоков между серверами в кластере также удорожает разработку.

В качестве перспективной альтернативы «традиционной» архитектуре была разработана архитектура, использующая преимущества функционального языка программирования Erlang. Erlang [3] – это язык параллельного программирования для разработки телекоммуникационных систем, предложенный компанией Ericsson. Наиболее примечательная особенность этого языка – концепция «легковесных процессов» - виртуальных сущностей, существующих лишь на уровне виртуальной машины Erlang, но не отражающихся на уровне ОС в виде процессов или потоков. Виртуальные процессы очень дёшевы – создание процесса требует лишь 4 микросекунды времени, а одна виртуальная машина может выполнять сотни тысяч виртуальных процессов [4]. Виртуальные процессы не имеют общих данных, а следовательно – не требуют синхронизации. Межпроцессная коммуникация возможна лишь на уровне т.н. «сообщений», пересылаемых одним процессом другому. Erlang также поддерживает встроенные средства распределённого программирования, что позволяет прозрачно масштабировать проектируемую систему на кластер серверов, не инвестируя ресурсы в собственную логику управления кластером. Разработка архитектуры на платформе Erlang, однако, также потребовала решения некоторых нетривиальных задач:

- Отсутствие общих данных не позволяет легко реализовать такие необходимые сущности, как общий счётчик производительности (количества обработанных всеми процессами сообщений в единицу времени);
- Erlang практически не предоставляет инструментов для создания пользовательских интерфейсов – в результате чего всю логику взаимодействия с пользователем пришлось реализовывать на других языках в виде отдельных компонентов;
- Управление столь большим количеством параллельно выполняющихся процессов требует эффективно организованной модели – в качестве таковой была выбрана модель «дерева управления процессами (supervision tree)" [5].

Приведённая ниже таблица сравнивает две рассмотренных архитектуры по критериям стоимости разработки и производительности полученного приложения (сравнение выполнялось на идентичных аппаратных конфигурациях: кластер из двух двухпроцессорных серверов Intel E5645, 16Гб оперативной памяти).

Архитектура	**Стоимость разработки (чел/дней)**	**Измеренная производительность (на один сервер в секунду)**		
		Получение СМС от клиента	**Биллинг и Статитика**	**Доставка СМС на Uplink**
многопоточная Java	**324**	780	370	1'100
Erlang/OTP	**290**	2'400	880	3'900

Полученные результаты позволяют смело утверждать, что, несмотря на некоторые сложности при создании архитектуры, предлагаемая платформой Erlang концепция легковесных независимых виртуальных процессов предоставляет весьма значительные преимущества при разработке приложений для нужд сетей сотовой связи.

Список использованных источников:

1. MobiThinking.com Mobile Marketing electronic magazine. – Mode of access: http://mobithinking.com/mobile-marketing-tools/latest-mobile-stats/c#mobilemessaging – Date of access: 22.01.2013
2. Goetz, Brian. Java Concurrency in Practice / Brian Goetz, with Tim Peierls – Pearson Education Inc, 2006
3. Joe Armstrong, Robert Virding, Claes Wikstrom, and Mike Williams. Concurrent Programming in Erlang, Second Edition. – Prentice-Hall, 1996.
4. Armstrong, Joe. Programming Erlang. Software for a Concurrent World. – The Pragmatic Bookshelf, 2007.
5. Erlang.org online documentation. – Mode of access: http://www.erlang.org/documentation/doc-4.9.1/doc/design_principles/sup_princ.html – Date of access: 22.01.2013