Мониторинг вызовов методов на удаленных объектах в распределенной вычислительной среде*

О. С. Естехин

Институт системного анализа Российской академии наук (ИСА РАН)

Проблемы мониторинга распределенных приложений

В процессе поиска и устранения ошибок, а так при оптимизации программного обеспечения с точки зрения скорости работы или требуемых ресурсов, встает задача трассировки хода выполнения программы. В это понятие, в частности, входят пошаговое выполнение инструкций, отслеживание последовательности вызова методов и значений переменных, прерывание программы при наступлении определенных условий. Существуют специализированные утилиты, позволяющие выполнять все эти действия на уровне машинного кода, кроме того большинство современных интегрированных сред разработки имеют в своем составе модуль для трассировки и отладки приложения на уровне исходного кода. Все эти утилиты отлично работают в случае, когда всё приложение целиком находится в одном адресном пространстве, то есть работает на одном компьютере в рамках одного процесса. Широкое распространение многозадачных операционных систем и многоядерных процессоров добавляет необходимость трассировки многопоточных приложений, но и эта задача успешно решается теми же способами, что и при трассировке однопоточного приложения.

* Работа выполнена при поддержке Президиума РАН (программа фундаментальных исследований 15П, проект 1.4).

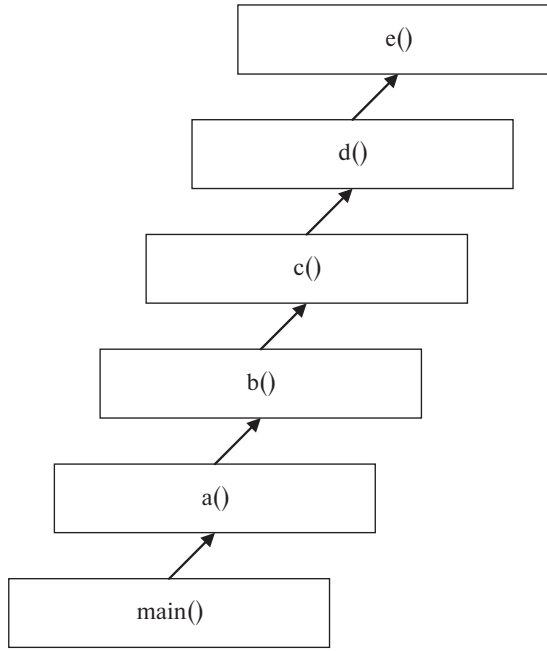


Рис. 4. Цепочка вызовов внутри одного процесса в одном адресном пространстве

Отличие распределенных приложений состоит в том, что они могут состоять из нескольких процессов, каждый из которых находится в отдельном адресном пространстве. Это может означать как запуск нескольких процессов на одном компьютере, так и использование нескольких компьютеров. Несмотря на то, что к каждому отдельному процессу можно применять обычные методы трассировки при помощи стандартных утилит, этого зачастую не достаточно для анализа работы всего распределенного приложения, как единого целого.

С точки зрения трассировки хода выполнения программы отличительными характеристиками распределенного приложения являются:

- Наличие нескольких одновременно работающих процессов. Процессы могут быть как гомогенными (то есть выполнять один и тот же набор операций), так и гетерогенными (то есть выполнять различные функции). Процессы могут запускаться и останавливаться независимо друг от друга.

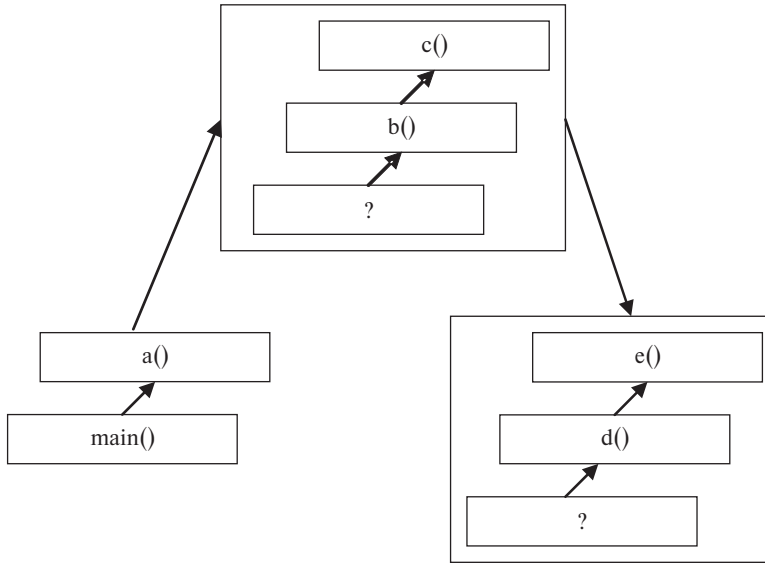


Рис. 5. Цепочка вызовов между процессами в разных адресных пространствах

- Асинхронная природа обмена информацией между процессами. В отличие от обычных приложений, вызов метода не всегда означает приостановку работы вызывающего метода. Вызываемые и вызывающий процессы могут находиться на разных компьютерах, что позволяет вызывающему процессу выполнять какую-либо другую работу во время передачи данных по сети и работы вызываемого процесса.
- Непредсказуемость момента вызова того или иного метода. Процесс может в любой момент получить запрос на выполнение какой-либо операции от другого процесса.
- Анонимность вызова. В обычном приложении в любой момент времени можно построить полную цепочку вызова методов. В распределенном приложении даже с использованием сложных схем авторизации зачастую можно проследить последовательность вызовов не более чем на один процесс назад. В случае сложных межпроцессорных взаимодействий между связанными по сети компонентами определить точную последовательность событий, приведших к вызову того или иного метода в определенном процессе, практически невозможно.

Эти и другие факторы приводят к тому, что при анализе работы распределенного приложения приходится использовать неудобные и устаревшие методики, например ведение журнала событий на каждом компоненте (например, в виде текстовых файлов с метками времени) и последующее сравнение данных, полученных на разных компьютерах, вручную. Даже с учетом того, что возможности современных компьютеров и средств виртуализации позволяют создать модель сети из нескольких компьютеров на одной физической машине, отладка распределенного приложения, как единого целого, является затруднительной задачей.

Мониторинг вызовов удаленных методов, основанный на сообщениях

Альтернативный подход заключается в создании специального трассировщика, который мог бы в режиме реального времени показывать текущие вызовы удаленных методов. С технической точки зрения для создания распределенного трассировщика необходимо обеспечить эффективный обмен сообщениями между компонентами распределенного приложения и трассировщиком, а так же решить задачу идентификации группы сообщений, как принадлежащих одному и тому же удаленному вызову. Минимально необходимый набор сообщений состоит из следующих команд:

- beforeInvoke — сообщение отсылается вызывающим процессом перед тем, как он вызовет удаленный метод.
- beforeServe — сообщение отсылается вызываемым процессом перед тем, как он начнет обрабатывать полученный запрос.
- afterServe — сообщение отсылается вызываемым процессом после того, как он закончил обрабатывать полученный запрос.
- afterInvoke — сообщение отсылается вызывающим процессом после того, как он получил результаты удаленного вызова.

При использовании подобного подхода для передачи сообщений на монитор необходимо использовать тот же механизм, который используется для удаленных вызовов между процессами. Это существенно ограничивает переносимость монитора между разными middleware платформами.

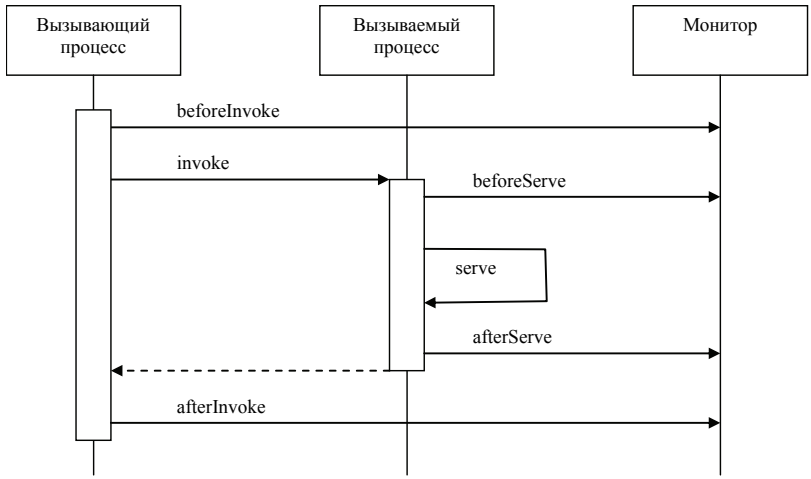


Рис. 6. Диаграмма последовательности вызовов с использованием трассировочных сообщений

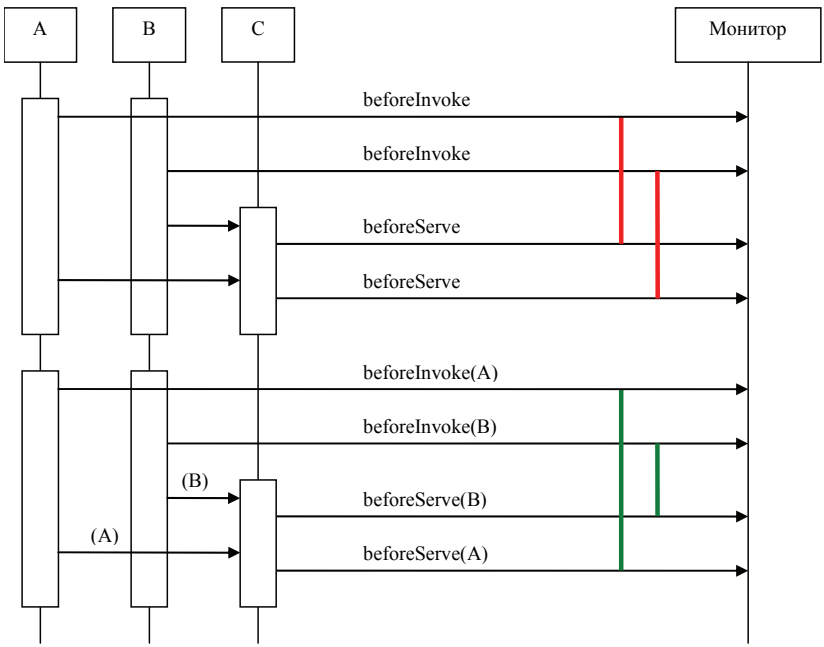


Рис. 7. Необходимость использования уникального идентификатора вызова

При мониторинге сложного распределенного приложения, в котором в каждый момент времени происходит большое количество удаленных вызовов, причем вызываться может один и тот же метод, необходимо в каждом трассировочном сообщении передавать идентификатор, позволяющий указать, к какому именно вызову относится это сообщение. В частности, это означает, что в рамках одного и того же вызова идентификаторы соответствующих beforeInvoke, beforeServe, afterServe и afterInvoke должны быть одинаковыми. С учетом того, что вызывающий процесс должен назначать новый уникальный идентификатор каждому вызову, он должен передавать этот идентификатор вызываемому процессу в дополнение к тем данным, которые требуются непосредственно для вызова удаленного метода. Так как сетевые задержки при передаче сообщений могут привести к тому, что сообщения от разных процессов придут на монитор в порядке, отличном от реального порядка выполнения, отсутствие уникального идентификатора вызова может привести к неправильной ассоциации между трассировочными сообщениями.

Необходимость передачи дополнительной метаинформации накладывает существенные ограничения на использование монитора, так как не все модели удаленных вызовов поддерживают подобную функциональность.

Реализация мониторинга удаленных вызовов на примере среды IARnet

Для проверки работоспособности предлагаемого подхода был разработан прототип сервиса для мониторинга удаленных вызовов в среде распределенных вычислений IARnet.

Среда IARnet базируется на промежуточном программном обеспечении ICE, которое позволяет передавать произвольную метаинформацию в контексте каждого удаленного вызова. Сервис мониторинга представляет собой удаленный объект, методы которого служат для получения трассировочных сообщений.

Определение интерфейса сервиса мониторинга на языке Slice приведено в следующем листинге:

```
module IARnet {
  module MonitoringService {
    interface Monitor {
```

```
    void beforeInvoke(string id, string remoteId,
      string operation, string unique);
    void afterInvoke(string id, string remoteId,
      string operation, string unique);
    void beforeServe(string id, string operation,
      string unique);
    void afterServe(string id, string operation,
      string unique);
   };
  };
};
```

Каждый метод принимает на вход идентификатор локального процесса, название операции, и уникальный идентификатор вызова. На вызывающей стороне дополнительно указывается идентификатор вызываемого процесса.

Пример использования монитора на вызывающей стороне на примере вызова метода add, складывающего два числа на вершине стека, приведен ниже:

```
String unique = Util.generateUUID();
 Map<String, String> context = new HashMap<String,
  String>();
 context.put("IARnet.MonitoringService.unique",
  unique);
 try {
  monitor.beforeInvoke(identity, remoteIdentity,
    "add", unique);
  calculator.add(context);
 } finally {
  monitor.afterInvoke(identity, remoteIdentity,
  "add", unique);
}
```

Соответствующая ему реализация на вызываемой стороне выглядит следующим образом:

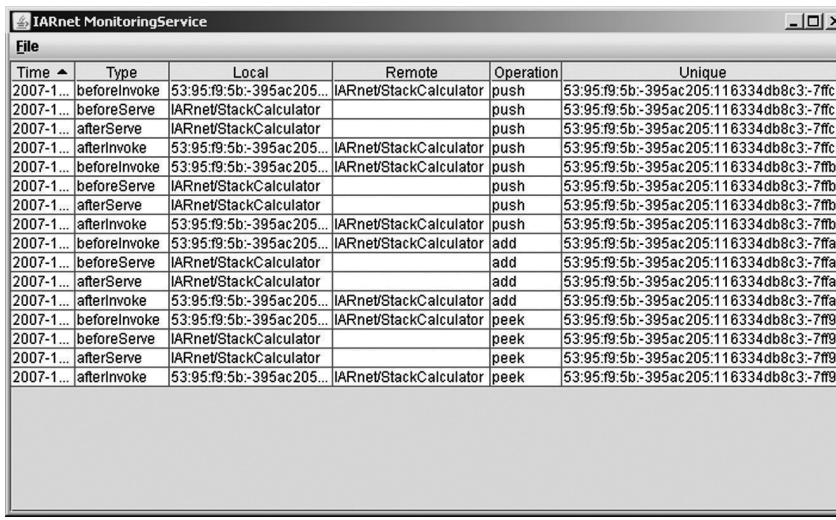
```
public void add(Current c) throws
 StackUnderflowException {
 String identity = Util.identityToString(c.id);
 String unique =
  c.ctx.get("IARnet.MonitoringService.unique");
 try {
  monitor.beforeServe(identity, "add", unique);
 if (stack.size() < 2) {
  throw new StackUnderflowException();
 }
```

```
        int b = stack.pop();
        int a = stack.pop();
        stack.push(a + b);
        } finally {
        monitor.afterServe(identity, "add", unique);
        }
    }
```

Ниже показан скриншот окна монитора после вызова методов, размещающих два числа на стеке, и затем выполняющих операции сложения и просмотра значения на вершине стека.

Разработанный сервис мониторинга показал, что трассировка вызова удаленных методов между границами процессов возможна, а получаемая в результате информация позволяет наглядно увидеть работу распределенного приложения на уровне, недоступном для обычных программ трассировки. В то же время, использование сервиса мониторинга выявило следующие недостатки предложенного подхода:

- Необходимость внесения дополнительных конструкций в исходный код программы. Отчасти этот недостаток можно преодолеть при помощи использования шаблона программирования Wrapper, который позволяет автоматизировать процесс общения с монитором.



Time ▲	Type	Local	Remote	Operation	Unique
2007-1...	beforeInvoke	53:95:f9:5b:-395ac205...	IARnet/StackCalculator	push	53:95:f9:5b:-395ac205:116334db8c3:-7ffc
2007-1...	beforeServe	IARnet/StackCalculator		push	53:95:f9:5b:-395ac205:116334db8c3:-7ffc
2007-1...	afterServe	IARnet/StackCalculator		push	53:95:f9:5b:-395ac205:116334db8c3:-7ffc
2007-1...	afterInvoke	53:95:f9:5b:-395ac205...	IARnet/StackCalculator	push	53:95:f9:5b:-395ac205:116334db8c3:-7ffc
2007-1...	beforeInvoke	53:95:f9:5b:-395ac205...	IARnet/StackCalculator	push	53:95:f9:5b:-395ac205:116334db8c3:-7ffb
2007-1...	beforeServe	IARnet/StackCalculator		push	53:95:f9:5b:-395ac205:116334db8c3:-7ffb
2007-1...	afterServe	IARnet/StackCalculator		push	53:95:f9:5b:-395ac205:116334db8c3:-7ffb
2007-1...	afterInvoke	53:95:f9:5b:-395ac205...	IARnet/StackCalculator	push	53:95:f9:5b:-395ac205:116334db8c3:-7ffb
2007-1...	beforeInvoke	53:95:f9:5b:-395ac205...	IARnet/StackCalculator	add	53:95:f9:5b:-395ac205:116334db8c3:-7ffa
2007-1...	beforeServe	IARnet/StackCalculator		add	53:95:f9:5b:-395ac205:116334db8c3:-7ffa
2007-1...	afterServe	IARnet/StackCalculator		add	53:95:f9:5b:-395ac205:116334db8c3:-7ffa
2007-1...	afterInvoke	53:95:f9:5b:-395ac205...	IARnet/StackCalculator	add	53:95:f9:5b:-395ac205:116334db8c3:-7ffa
2007-1...	beforeInvoke	53:95:f9:5b:-395ac205...	IARnet/StackCalculator	peek	53:95:f9:5b:-395ac205:116334db8c3:-7ff9
2007-1...	beforeServe	IARnet/StackCalculator		peek	53:95:f9:5b:-395ac205:116334db8c3:-7ff9
2007-1...	afterServe	IARnet/StackCalculator		peek	53:95:f9:5b:-395ac205:116334db8c3:-7ff9
2007-1...	afterInvoke	53:95:f9:5b:-395ac205...	IARnet/StackCalculator	peek	53:95:f9:5b:-395ac205:116334db8c3:-7ff9

Рис. 8. Пример работы монитора

- Существенное уменьшение скорости работы. Без учета времени, требуемого на обработку вызова на принимающей стороне, каждый удаленный вызов требует отправки четырех сообщений на сервис мониторинга. То есть в случаях, когда сетевые задержки сопоставимы со временем непосредственной обработки запроса, общее время вызова удаленного метода увеличивается примерно в пять раз.

Выводы

Проведенное исследование и реализованный прототип сервиса мониторинга показали, что трассировка вызовов удаленных методов может иметь практическую пользу при разработке и отладке распределенных приложений. В отличие от обычных программ-трассировщиков, привязанных к языку программирования или архитектуре центрального процессора, распределенный трассировщик привязан к промежуточному программному обеспечению, используемому для организации удаленных вызовов. Наиболее приемлемым вариантом для реализации распределенного трассировщика была бы возможность подключения плагина, получающего уведомления о начале и окончании каждого удаленного вызова, к каждому отдельному процессу в составе распределенного приложения, но, к сожалению, подобная функциональность отсутствует в большинстве промежуточного программного обеспечения.

Литература

1. *Волошинов В. В., Естехин О. С., Сухорослов О. В.* Программная архитектура и принципы реализации системы IARnet // Первая международная конференция «Системный анализ и информационные технологии» САИТ-2005 (12–16 сентября 2005 г., Переславль-Залесский, Россия): Труды конференции. В 2 т. Т. 2. М.: КомКнига/URSS, 2005. С. 141–146.
2. *Афанасьев А. П., Волошинов В. В., Сухорослов О. В.* Распределенная информационно-алгоритмическая среда для решения научных задач // Труды III Международной конференции «Параллельные вычисления и задачи управления» PACO-2006 памяти И. В. Прангишвили. Москва, 2–4 октября 2006 г. Институт проблем управления им. В. А. Трапезникова РАН. М.: Институт проблем управления им. В. А. Трапезникова РАН, 2006. С. 866–879.
3. *Michi Henning, Mark Spruiell.* Distributed Programming with Ice. Revision 3.2.1, August 2007 / http://zeroc.com/doc/Ice-3.2.1/manual