Распределённые объектные технологии: **Акторная модель и Erlang/OTP**

Д. А. Усталов

УрФУ и ИММ УрО РАН

24 мая 2016 г.

Оценка доступности систем

Уровень доступности («девятки») — отношение времени доступности системы к общему времени её функционирования.

Встречается в соглашениях об уровне обслуживания (SLA):

- $90\ \%$ — одна «девятка», $\approx 36{,}5$ **сут.** простоя в год;
- $99\ \%$ — две «девятки», $\approx 3{,}65$ сут. простоя в год;
- $99{,}9\ \%$ — три «девятки», $\approx 8{,}76$ **час.** простоя в год;
- $99{,}99\ \%$ — четыре «девятки», $\approx 52{,}56$ **мин.** простоя в год;
- $99{,}9999\ \%$ — шесть «девяток», $\approx 31{,}5$ **сек.** простоя в год.

Примеры: Amazon, Azure и Google ($99{,}95\ \% \approx 4{,}38$ час./год).

Высокодоступные системы

- Строить распределённые системы тяжело.
- Строить высокодоступные системы тяжело.
- Как строить высокодоступные распределённые системы?

Случай в Ericsson

В 90-е гг. компания Ericsson разрабатывала высокоскоростной коммутатор пакетов, передаваемых в асинхронном режиме.

- В предыдущих продуктах компании использовался **Prolog**, который перестал соответствовать требованиям.
- С учётом имеющегося опыта и требований создан язык программирования и платформа для построения высокодоступных систем.
- Позже, в 1998 г., технология под названием Erlang/OTP стала открытой.

Язык программирования Erlang

- Функциональный язык программирования с динамической типизацией.
- Разработан в 1986 г. на основе **Prolog**.
- Компилируется в байт-код для запуска в виртуальной машине.
- Содержит удобные инструменты построения распределённых систем.

http://erlang.org/faq/getting_started.html

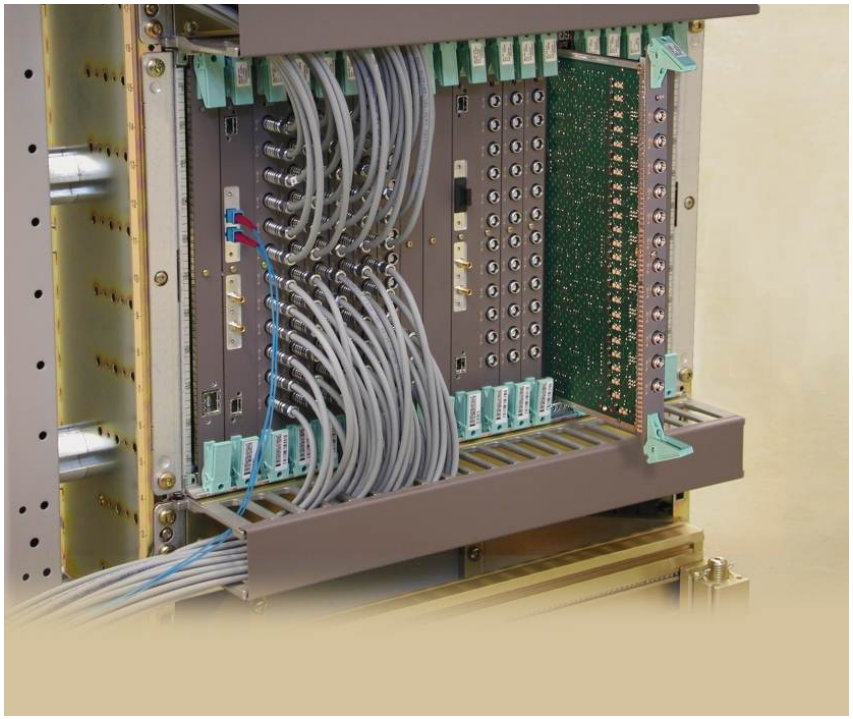
Open Telecom Platform (OTP)

Open Telecom Platform включает в себя:

- интерпретатор и компилятор Erlang;
- протокол обмена данными между узлами;
- брокер CORBA;
- статический анализатор кода;
- распределённую СУБД Mnesia;
- стандартную библиотеку.

Ericsson AXD301

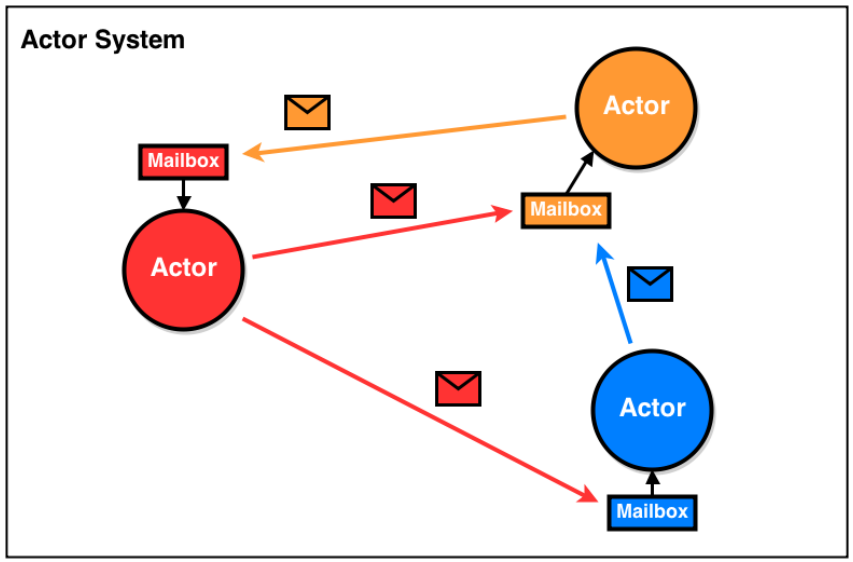
- Около миллиона строк кода на языке Erlang.
- Девять «девяток», т. е. уровень доступности $99{,}9999999\ \%$.
- Использование **акторной модели**.



http://www.erlang.se/.../ericsson_review_axd301_1998012.pdf

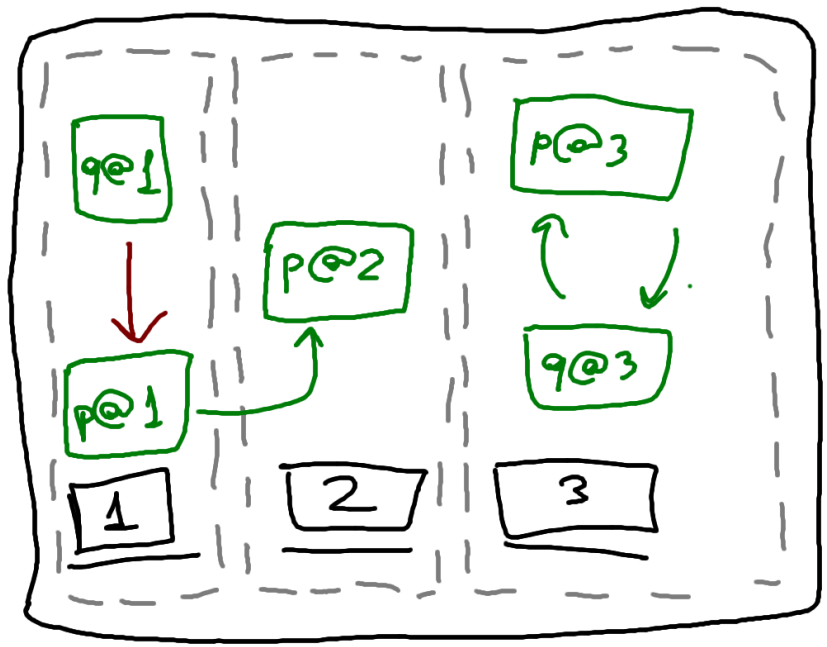
Акторная модель

- **Актор** — сущность, способная обмениваться сообщениями и реагировать на них.
- Как правило, акторы работают в асинхронной параллельной среде — **акторной системе** без разделяемого состояния.
- Отказоустойчивость обеспечивается иерархиями акторов: принцип «**пускай падает**».



Сетевая прозрачность

- OTP обеспечивает сетевую прозрачность в виде mesh-сети.
- При вызове `net_kernel` новый узел добавляется в mesh.
- Узлы имеют имя вида `sname@host`, для сетевого подключения требуются «куки».
- Имеется поддержка SSL.



Демонстрация работы: печать текста

```
#!/usr/bin/env escript

main(_) ->
  io:fwrite("Hello!~n").
```

В учебных примерах можно запускать программы при помощи `escript` (пожалуйста, не делайте так на работе).

Демонстрация работы: факториал

```
#!/usr/bin/env escript

fac(0) -> 1;
fac(N) -> N * fac(N - 1).

main(_) ->
  N = 1337,
  io:fwrite("Factorial of ~w is ~w~n", [N, fac(N)]).
```

Демонстрация работы: процессы

```
#!/usr/bin/env escript

f(String) -> io:fwrite("~s~n", [String]).

main(_) ->
  spawn(fun() -> f("hello") end),
  spawn(fun() -> f("world") end),
  timer:sleep(1000).
```

Демонстрация работы: идентификаторы процессов

```
#!/usr/bin/env escript

f() ->
  receive
    String ->
      io:fwrite("~s~n", [String]),
      f()
  end.

main(_) ->
  Writer = spawn(fun() -> f() end),
  Writer ! "hello",
  Writer ! "world",
  timer:sleep(1000).
```

Демонстрация работы: регистрация процессов

```
#!/usr/bin/env escript

f() ->
  receive
    String ->
      io:fwrite("~s~n", [String]),
      f()
  end.

main(_) ->
  register(writer, spawn(fun() -> f() end)),
  writer ! "hello",
  writer ! "world",
  timer:sleep(1000).
```

Демонстрация работы: сетевая прозрачность

```
$ erl -name vm1@172.19.132.6 -setcookie doturfu
$ erl -name vm2@172.19.3.176 -setcookie doturfu

1> net_kernel:connect('vm1@172.19.132.6').
2> spawn('vm1@172.19.132.6', fun() ->
 io:format("I'm on ~p~n", [node()]) end).
3> register(shell, self()).
4> {shell, 'vm1@172.19.132.6'} ! {hello, from, self()}.
5> receive {hello, from, OtherShell} ->
 OtherShell ! <<"hey there!">> end.
6> flush().
```

http://learnyousomeerlang.com/distribunomicon

Обсуждение

Использование: Facebook и Whatsapp, RabbitMQ и ejabberd, Riak, CouchDB и Amazon SimpleDB, и т. д.

- ✚ Наличие готовых элементов для построения распределённых систем.
- ✚ Принцип «пускай падает» за счёт лёгких процессов.
- ✚ Отсутствие разделяемого состояния.
- ✚ Масштабируемость и эффективная работа с сетью.
- ➖ Местами недостаточно развитая экосистема.
- ➖ Некоторая ограниченность применения из-за особенностей виртуальной машины и синтаксиса (хотя см. Elixir).

Доступна реализация акторной модели для разных языков программирования: Akka (JVM), Orleans (.Net), Celluloid (Ruby), Pykka (Python), и т. д.

Домашнее задание

Построить акторную систему для подсчёта размера файлов в текущей директории по принципу, близкому к MapReduce.

- Программа *client* передаёт список файлов в процесс *master*, получает и выводит результат работы.
- Программа *master* запускает одноимённый процесс, который создаёт и запоминает несколько процессов *worker*.
- Процесс *master* при получении списка файлов распределяет задания между процессами *worker*, ожидает выполнения работ, возвращает клиенту финальный результат.
- Каждый процесс *worker* получает задание от *master*, выполняет его и возвращает ответ.
- Два вида заданий: *map* — определить размер указанного файла и *reduce* — вычислить сумму списка чисел.
- Выбор языка программирования произволен.

Спасибо за внимание!

Вопросы?

Дмитрий Усталов

https://linkedin.com/in/ustalov

http://kvkt.urfuclub.ru/courses/dot/

https://telegram.me/doturfu

dmitry.ustalov@urfu.ru

Это была последняя тема.