Важные особенности Эрланг

Concurrency

Fault Tolerance

Distribution

Hot Code Upgrade

Важные особенности Эрланг

Symmetric Multiprocessing

Actor Model

Soft Real Time

Garbage Collection

Erlang Shell

Tracing

Важные особенности Эрланг

Erlang Run-Time System

ERTS

Concurrency

процессы являются базовой сущностью языка

процессы легковесны,
их можно создавать десятки и сотни тысяч

переключение между процессами очень быстрое

Concurrency

нет разделяемой области памяти,
каждый процесс имеет свою изолированную память

ошибки в процессах также изолированы,
падение одного процесса не влияет на работу остальных

Concurrency

Данные между процессами передаются путем "отправки сообщений" (message passing)

при этом данные копируются из памяти одного процесса в память другого

Concurrency

Ограничение на количество процессов:

минимум 1024

максимум 134,217,727 (2^27)

по умолчанию 262,144 (2^18)

Concurrency

Запуск нового процесса – 3-5 микросекунд

Concurrency

Память процесса:

стэк, куча, почтовый ящик, метаданные

на старте 2696 байт

Concurrency

запускается несколько планировщиков,
соответственно количеству процессорных ядер

каждый планировщик использует один процесс ОС,
и поверх него запускает эрланговские процессы

Concurrency

Lukas Larsson

Understanding the Erlang Scheduler

https://www.youtube.com/watch?v=tBAM_N9qPno

Fault Tolerance

Первый уровень защиты:

try .. catch

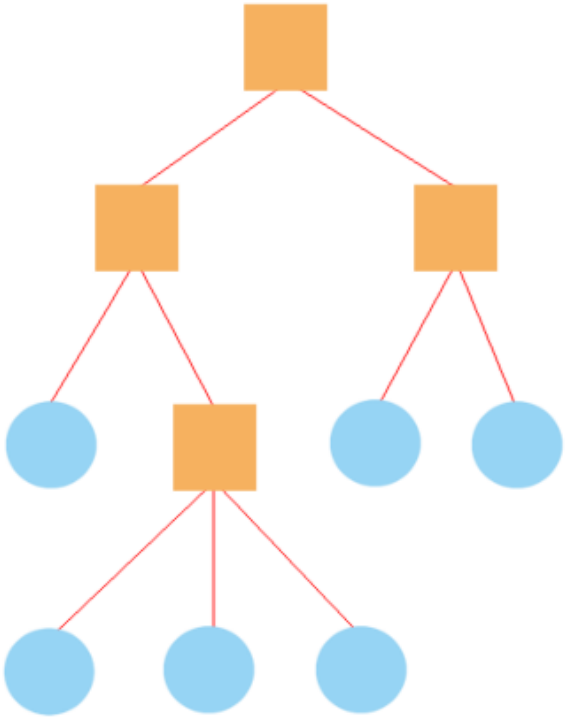
Fault Tolerance

Второй уровень защиты:

supervisor .. worker

Fault Tolerance

супервизоры и воркеры формируют дерево



Fault Tolerance

Третий уровень защиты:

объединение узлов в кластер

Distribution

Устойчивость к аппаратным авариям
является одним из требований к эрланг

Обеспечить эту устойчивость
можно только в распределенной системе

Distribution

Если инфраструктура состоит из сотен серверов,
то отказ железа случается регулярно,
и является штатной ситуацией

Distribution

Сетевая прозрачность

location transparency

Distribution

Сетевая прозрачность

касается не только отправки сообщений,

но и мониторинга процессов

супервизор может запускать и мониторить

воркер на другом узле

Distribution

эрланг-узлы, собранные в кластер,

формируют доверенную среду

(trusted environment)

Hot Code Upgrade

Можно загрузить в рантайм новую версию кода
и переключить выполнение процесса
со старой версии на новую,
сохранив состояние его памяти

Hot Code Upgrade

Если в коде изменились структуры данных,
то есть способ мигрировать данные
из старой структуры в новую

Symmetric Multiprocessing

Эрланг умеет эффективно использовать
все процессорные ядра в системе,
и перераспределять нагрузку между ними

Symmetric Multiprocessing

Проверено на чипах с 1024 ядрами

Actor Model

Система состоит из акторов,
которые действуют параллельно
и независимо друг от друга

Actor Model

Акторы общаются друг с другом
с помощью отправки сообщений
(message passing)

Actor Model

Отправка сообщений является асинхронной

Actor Model

Каждый актор имеет mailbox,
где накапливаются полученные им сообщения

Actor Model

Популярна и в других языках

библиотека Akka

http://akka.io/

для Scala и Java

Soft Real Time

Это возможно благодаря:

вытесняющей многозадачности (preemptive scheduling)

настраиваемому IO

особенностям сборки мусора (garbage collection)

Garbage Collection

Generational Copying garbage collection

два поколения объектов: молодые и старые

молодые чистятся часто, старые редко

Garbage Collection

отдельный сборщик мусора для каждого процесса

все они работают независимо друг от друга,
в разные моменты времени,
и останавливают только свой процесс

Garbage Collection

Не бывает эффекта **stop world**,

когда сборщик мусора должен остановить весь узел

для своей работы

Garbage Collection

Для короткоживущих процессов

GC просто не успевает запуститься

Для долгоживущих, но потребляющих мало памяти

(supervisor или какой-нибудь управляющий процесс),

GC не запускается, за ненадобностью

Garbage Collection

В результате сборка мусора

оказывает мало влияния

на общую производительность системы

Erlang Shell

REPL консоль

Read Eval Print Loop

Erlang Shell

Можно подключиться к работающему узлу,

вызывать любую функцию любого модуля,

отправить сообщение любому процессу,

прочитать и изменить состояние любого процесса

То есть, взаимодействовать с узлом в реальном времени

Tracing

механизм трассировки

встроен в виртуальную машину на низком уровне

мало влияет на производительность системы

Tracing

Можно получать в реальном времени данные:

жизненный цикл процессов
(запуск, остановка, связи с другими процессами)

отправка и получение сообщений

Tracing

Можно получать в реальном времени данные:

вызовы функций, их аргументы и возвращаемые значения

откуда вызывалась функция

Tracing

Можно получать в реальном времени данные:

информацию о работе планировщика

информацию о потреблении памяти
и работе сборщиков мусора

Tracing

Можно читать и менять содержимое памяти работающих процессов

Tracing

Можно узнать почти все о работе узла.

Сложность в том, чтобы узнать именно то, что нужно :)