Computación distribuida sobre Ruby on Rails

I Conferencia Rails Hispana

Madrid, Noviembre 2006

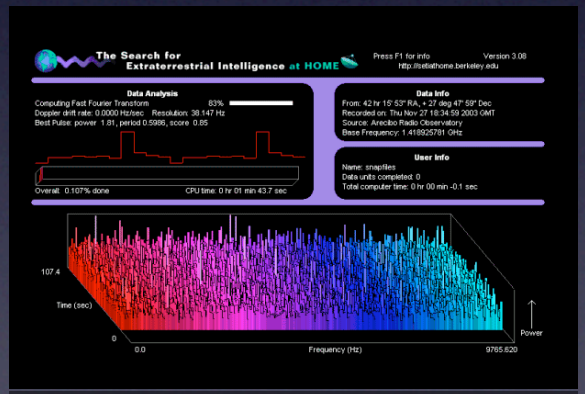
JJ Merelo
Dpto ATC, UGR

Juan Lupión
sobrerailes.com

- Computación distribuida
- Rails, AJAX y JSON
- Prueba de concepto: DCoR
- Resultados

El problema

- Ciclos de CPU desperdiciados en la mayoría de equipos
- Programas DC establecen redes a nivel de aplicación para aprovecharlos
    - gratuitas
    - comerciales
    - sigilosas
- Cada solución requiere instalar un cliente



El cliente que no se instala

- Porque ya viene instalado
- No hay ordenador sin navegador
    - ni móvil, ni set-top box, ni...
- El navegador es un cliente con capacidad de cómputo nativo
    - no Java, sino ECMAscript
    - cliente conectado
    - independiente del sistema operativo

- Computación distribuida
- Rails, AJAX y JSON
- Prueba de concepto: DCoR
- Resultados

¿Por qué Rails?

¿Por qué Rails?

- Si el cliente es un navegador web, necesitamos un *framework* web

¿Por qué Rails?

- Si el cliente es un navegador web, necesitamos un *framework* web
- Rails MOLA

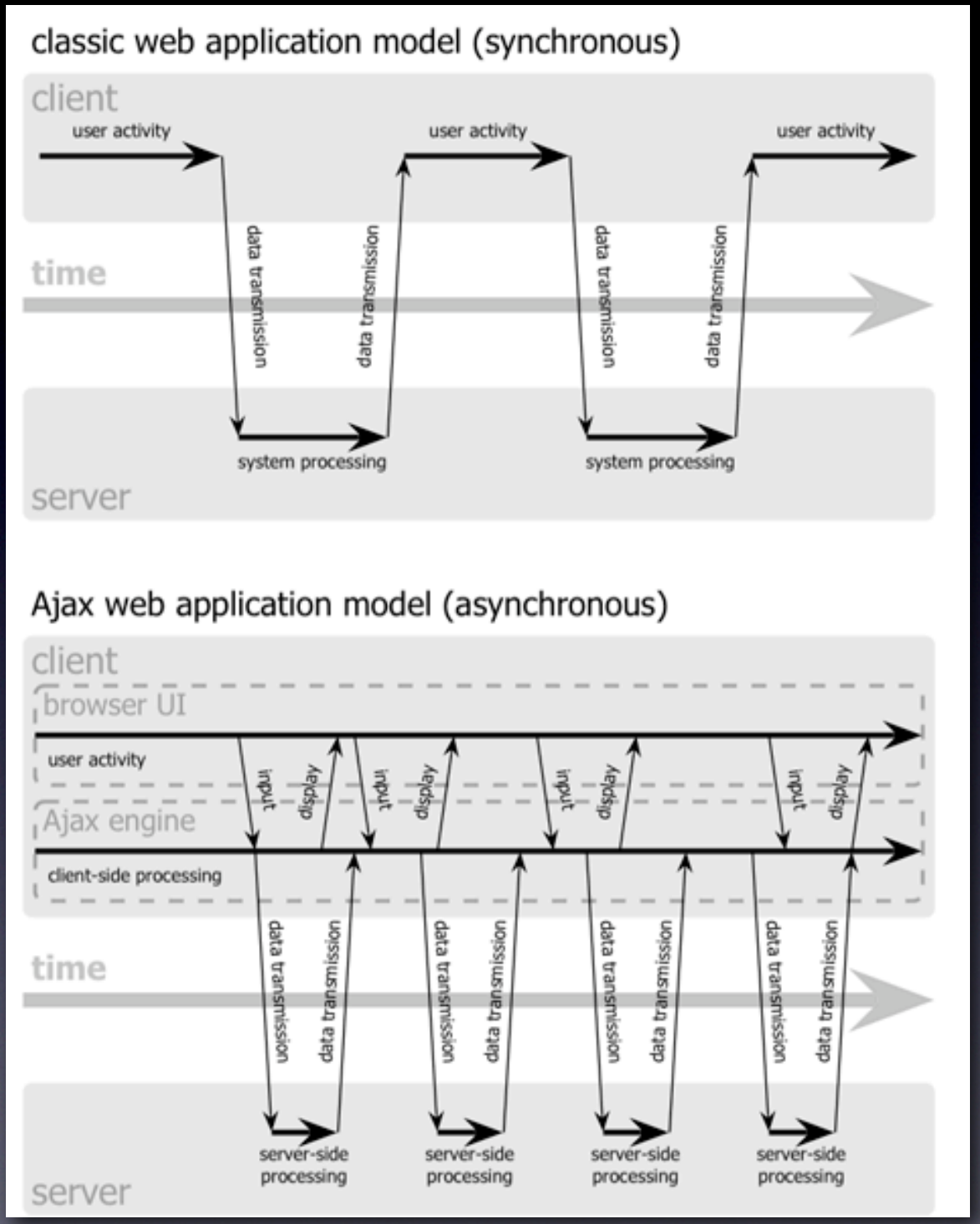
¿Por qué Rails?

- Si el cliente es un navegador web, necesitamos un *framework* web
- Rails MOLA
- ... esto no es la Conferencia Java o la Conferencia PHP

AJAX

- Sólo Javascript no es suficiente
- `XMLHttpRequest`
- Modelo de objetos compatible
  - más compatible gracias a Prototype
- Permite implementar un protocolo tipo RPC

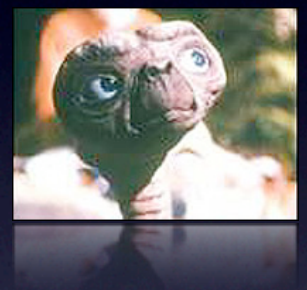
síncrono
vs
asíncrono



Representación de datos

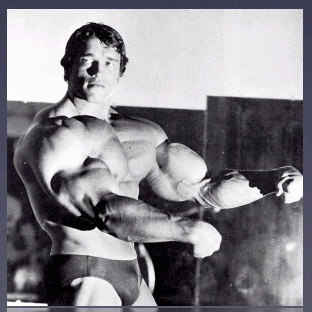
- Los locos 80
  - ASN1, Corba IDL

    binaria



- Los 90 *puntocom*
  - XML

    textual, pesada



Be JSON, my friend

JavaScript Object Notation

- Es Javascript
- Legible
- No necesita librerías en el navegador
- Eficiente

- Puede representar estructuras complejas
  - escalares, listas, diccionarios y combinaciones no recursivas
- Rails lo soporta: `to_json`

¡JSON ya!

```
>> a = { :abcde => [1, 2, [3,4], { :a => 1, :b => 2},
"11010101"], :fghij => { 3987135 => [1,2,3,4], 9827 =>
"00505" } }

=> {:abcde=>[1, 2, [3, 4], {:a=>1, :b=>2}, "11010101"], :fghij=>
{3987135=>[1, 2, 3, 4], 9827=>"00505"}}

>> a.to_json

=> "{\"abcde\": [1, 2, [3, 4], {\"a\": 1, \"b\": 2}, \"11010101
\"], \"fghij\": {3987135: [1, 2, 3, 4], 9827: \"00505\"}}"
```

- Computación distribuida
- Rails, AJAX y JSON
- Prueba de concepto: DCoR
- Resultados

¿Qué es DCoR?

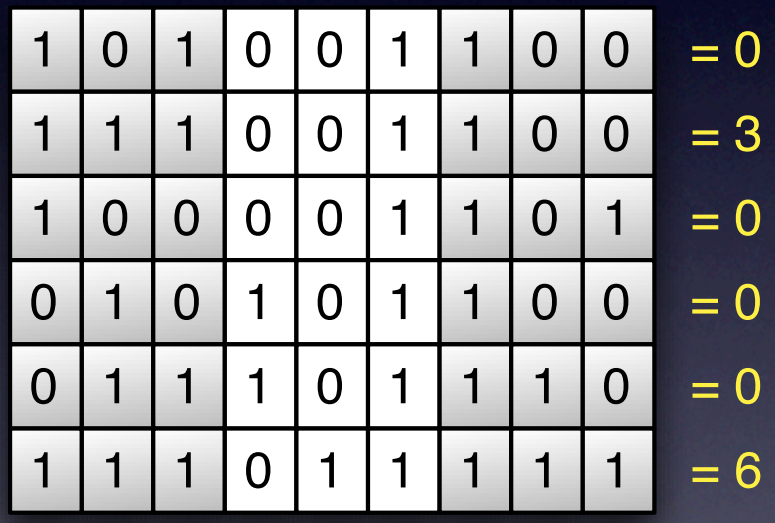
- Un sistema de computación distribuida que usa AJAX sobre RoR para problemas de computación evolutiva
- La distribución es a varios niveles
    - cliente web / servidor Rails
    - servidor Rails / servidor BD

Un algoritmo genético básico

I. Generar población inicial aleatoria
II. Mientras no hayamos terminado
    - evaluar aptitud de individuos
    - escoger individuos a alterar
    - aplicar operadores genéticos
    - eliminar los menos aptos, mezclar los nuevos

```
create_table "guys", :force => true do |t|
  t.column "cromosoma", :string
  t.column "fitness", :float
  t.column "algoritmo_id", :integer
  t.column "status", :integer
end
```

El problema *Royal Road*



1	0	1	0	0	1	1	0	0	= 0
1	1	1	0	0	1	1	0	0	= 3
1	0	0	0	0	1	1	0	1	= 0
0	1	0	1	0	1	1	0	0	= 0
0	1	1	1	0	1	1	1	0	= 0
1	1	1	0	1	1	1	1	1	= 6

```
function fitness ( str ) {
  var fitness = 0;
  var blockSize = 3;

  for ( var i = 0; i < str.length / blockSize; i++ ) {
    var block = new Boolean( true );

    for ( var j = 0; j < blockSize; j++ )
    {
      block = block &&
        (str[i*blockSize+j]=='1')?true:false;
    }

    if ( block )
    {
      fitness += blockSize;
    }
  }
  return fitness;
}
```

Operadores genéticos

1	1	1	0	1	1	1	1	1	x
1	1	1	0	0	1	1	0	0	y

1	0	1	0	1	1	1	1	1	mutación(x)
1	1	1	0	1	1	1	0	0	cruce(x,y)
1	1	1	0	0	1	1	1	1	mutación(x)
1	1	1	0	0	0	1	0	0	mutación(y)
1	1	1	0	1	1	1	1	1	cruce(x,y)
0	1	1	0	0	1	1	0	1	flip(y)

DCoR en el navegador

algoritmo.rhtml

```
<%= javascript_include_tag "fitness.js"  %>
<%= javascript_include_tag :defaults %>

<%= @content_for_layout %>
```

application.js

```
function getGeneration() {
}

function sendGeneration() {
}

function populationReceived() {
}

function resultSent() {
}
```

fitness.js

```
function fitness (str) {
      .....
}
```

DCoR en el navegador

algoritmo.rhtml

```
<%= javascript_include_tag "fitness.js"  %>
<%= javascript_include_tag :defaults %>

<%= @content_for_layout %>
```

application.js

```
function getGeneration() {
}

function sendGeneration() {
}

function populationReceived() {
}

function resultSent() {
}
```

fitness.js

```
function fitness (str) {
      .....
}
```

DCoR: Javascript

```
function getGeneration()
{
   new AJAX.Request("algoritmo/population" {
     asynchronous: true,
     method: "get",
     onSuccess: function (request) {
       populationReceived(request.responseText);
     }
    });
}
```

DCoR: Javascript

```
function getGeneration()
{
   new AJAX.Request("algoritmo/population" {
     asynchronous: true,
     method: "get",
     onSuccess: function (request) {
       populationReceived(request.responseText);
     }
    });
}
```

```
function populationReceived (str)
{
     var json_data = eval ("(" + str + ")") ;
     var myGuys = json_data.population;

     for (i=0; i<myGuys.length; i++)
     {
       myGuys[i].attributes.fitness =
           fitness(myGuys[i].attributes.cromosoma);
     }

     cadena = (myGuys.toJSONString());
     updateGuysDiv(myGuys);
     sendGeneration (cadena, myAlgorithmId)
}
```

DCoR: Javascript (y II)

```
function sendGeneration(str)
{
   new AJAX.Request ("/algoritmo/populationReady" {
     asynchronous: true,
     method: "post",
     parameters: "datos=" + cadena,
     onSuccess: function (request) {
         resultSent();
     }
     });
}
```

DCoR: Javascript (y II)

```
function sendGeneration(str)
{
   new AJAX.Request ("/algoritmo/populationReady" {
     asynchronous: true,
     method: "post",
     parameters: "datos=" + cadena,
     onSuccess: function (request) {
         resultSent();
     }
     });
}
```

```
function resultSent (str)
{
   getGeneration()
}
```

Controlador

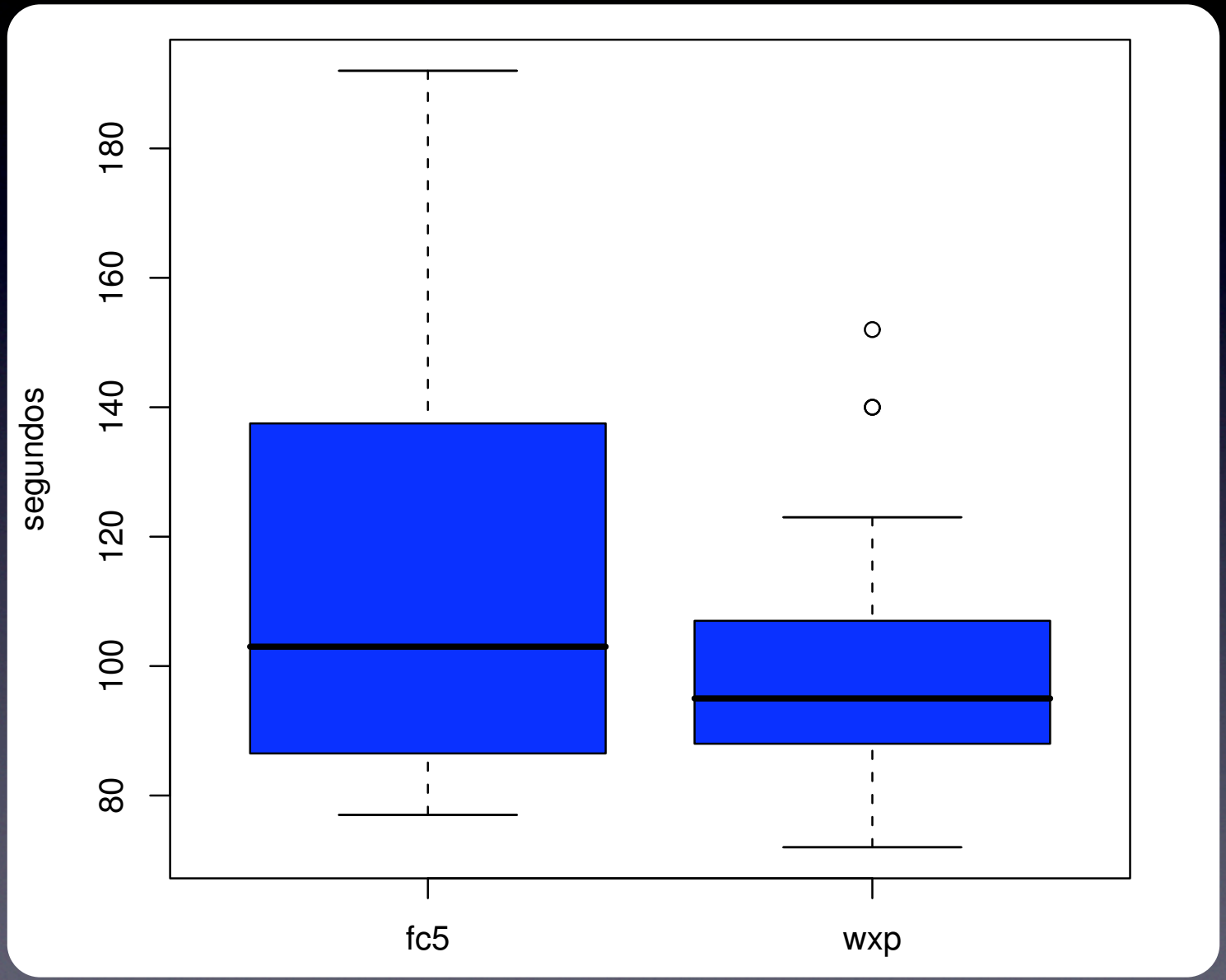
```
def population
    algoritmo = Algoritmo.find_by_id(params[:id])
    guys = algoritmo.find_guys_to_send

    if algoritmo.status == Algoritmo::RUN then

      Guy.update(guys.map { |guy| guy.id} , { :status =>  Guy::GUY_AWAITING_FITNESS} )

      statusdata = {
        "algorithm_id" => algoritmo.id,
        "target_fitness" => algoritmo.target_fitness,
        "generation" => algoritmo.generacion,
        "mutations" => algoritmo.mutations_count,
        "crossovers" => algoritmo.crossovers_count,
        "flipations" => algoritmo.flipations_count,
        "total_guys" => algoritmo.total_guys_generated
      }

      info_packet = {
        "status" => statusdata,
        "population" => guys
      }

      render :text => info_packet.to_json
    else
        render :update do |page|
        page.redirect_to(
          :controller=> 'algoritmo',
          :action =>'finish')
      end
    end
end
```

Controlador (y II)

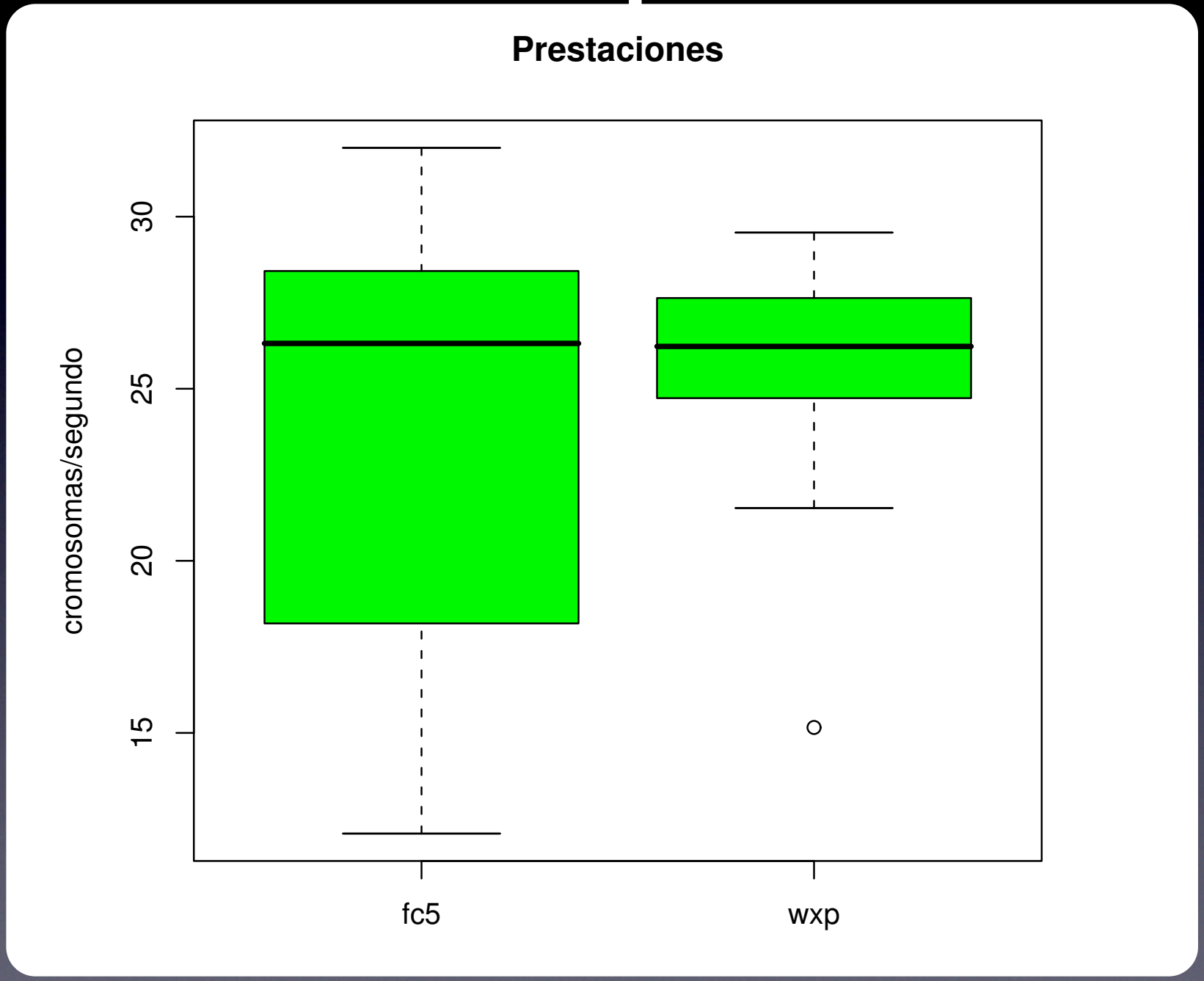
```
def populationReady

  data = parse_json(params[:datos])

  len = data.length

  guys_received = []

  for i in 0..(len-1)
    ind          = data[i]["attributes"]
    el_id        = ind["id"].to_i
    el_cromosoma = ind["cromosoma"]
    el_fitness   = ind["fitness"].to_f

   guys_recibidos <<
      update_guy(el_id, el_cromosoma, el_fitness,    Guy::GUY_FITNESS_AVAILABLE)
  end

  tournament(guys_recibidos)

  if (@algoritmo.total_guys_generated >= @algoritmo.max_guys) then
    @algoritmo.status = "FINISH"
  end

  @algoritmo.save
end
```

- Computación distribuida
- Rails, AJAX y JSON
- Prueba de concepto: DCoR
- Resultados

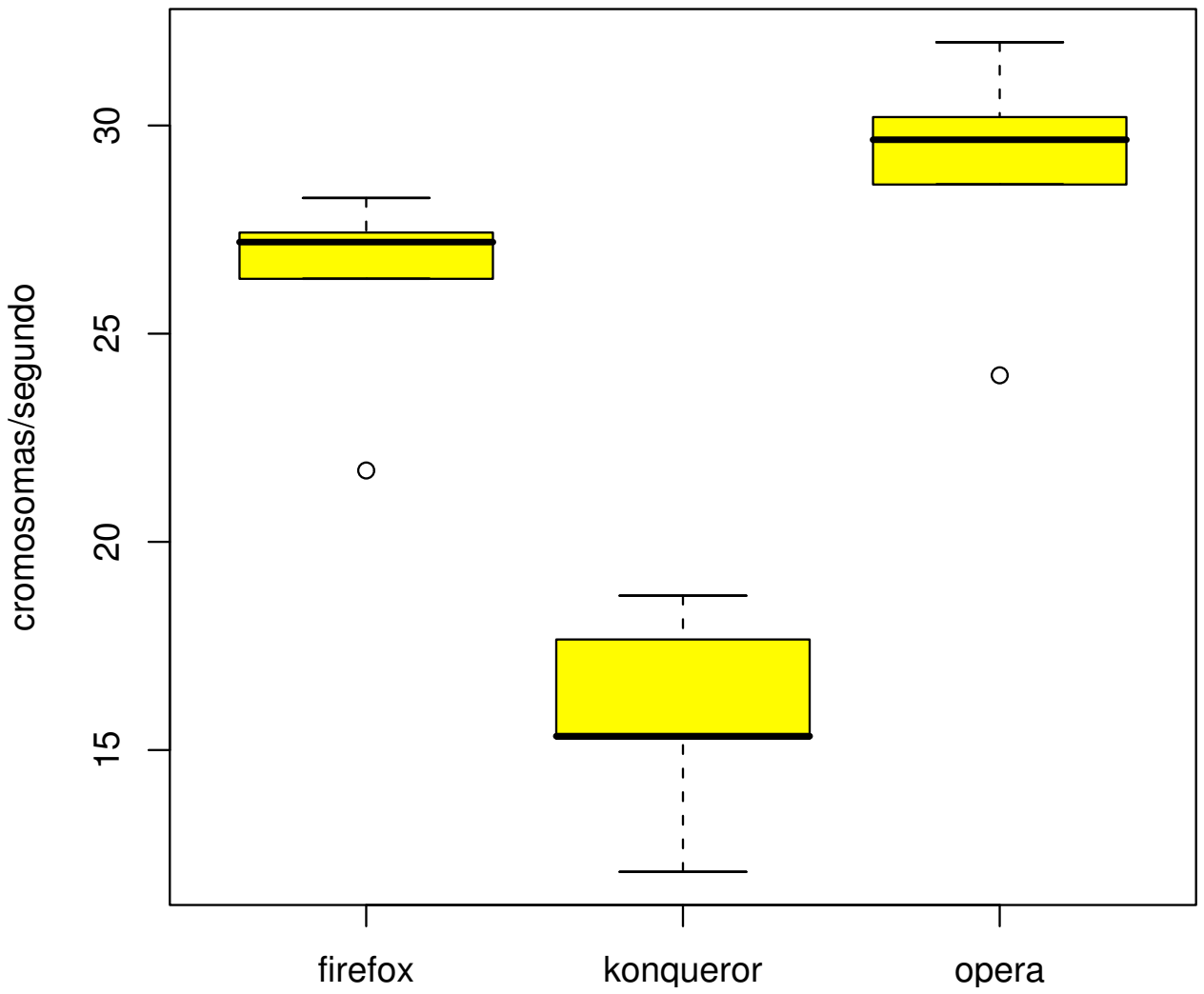
Servidores



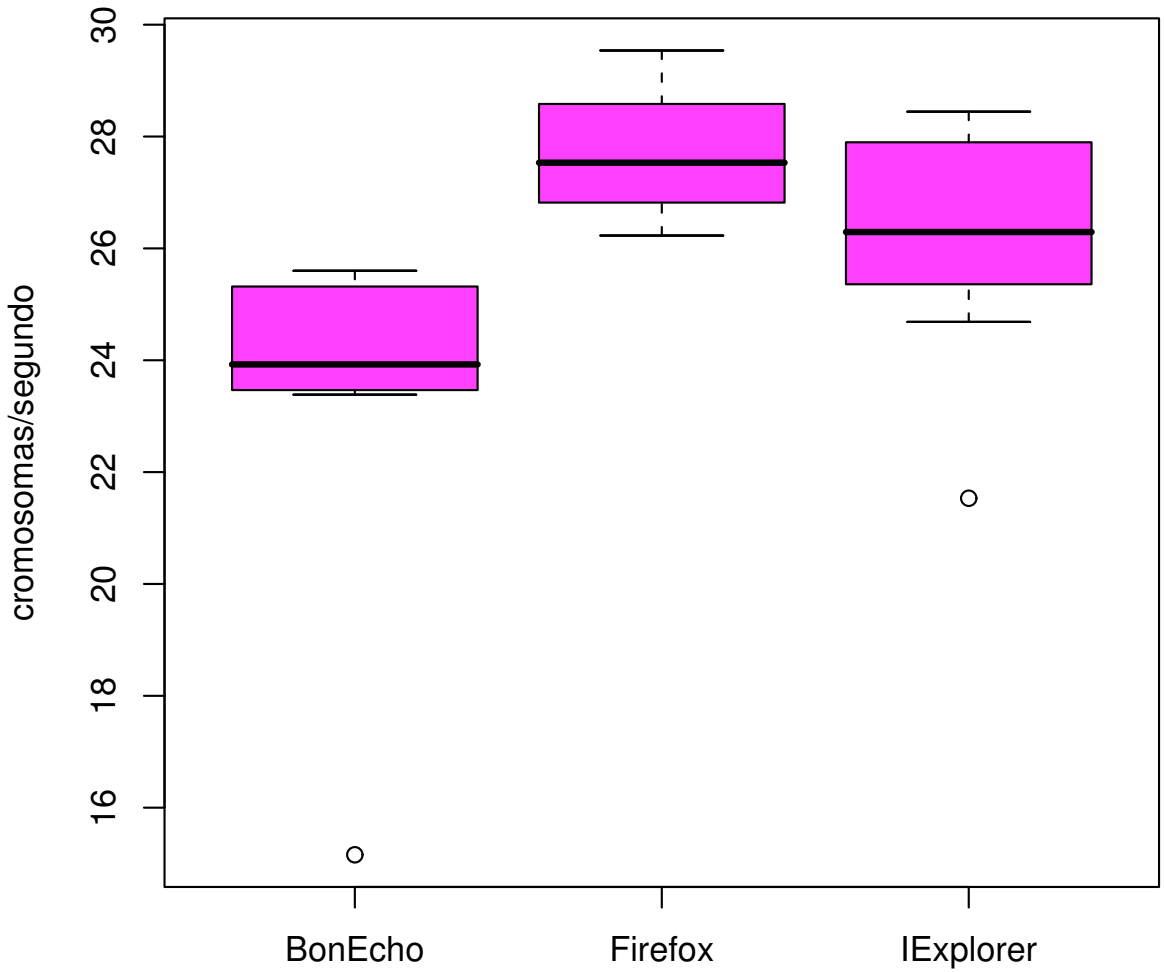
Sistemas operativos



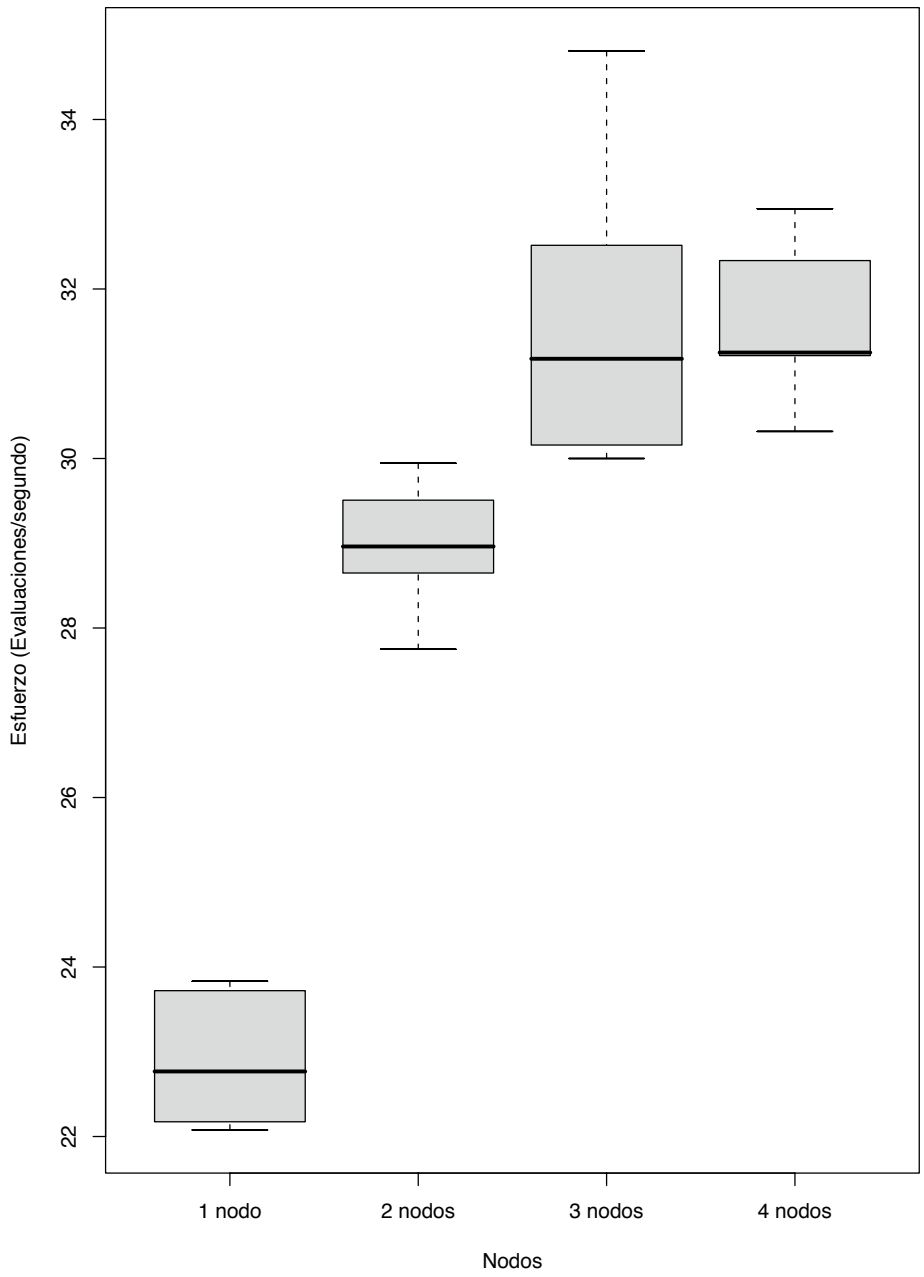
Prestaciones por navegador



Prestaciones por navegador (y II)



Escalado


Esfuerzo (Evaluaciones/segundo)
22
24
26
28
30
32
34
1 nodo
2 nodos
3 nodos
4 nodos
Nodos

Demo

http://localhost:3000

para saber más...

http://dconrails.rubyforge.org