Doloto:

Code Splitting for Web 2.0 Applications
Web 2.0 is Upon Us
Web 1.0 → Web 2.0

- **Server-side computation**
- **Static HTML**
- **Client-side rendering**
Distributed Applications

Server

Clients

Run your code here
Catch-22

Move code to client for responsiveness

Execution can’t start without the code
A Web 2.0 Application Disected

- Talks to 14 backend services (traffic, images, directions, ads, ...)
- 70,000+ lines of JavaScript code downloaded
- 2,855 Functions
- 1+ MB code
Motivation for Doloto

- **Idea behind Doloto**
  - Start with a small piece of code on the client
  - Download required code on demand (pull)
  - Send code when bandwidth available (push)

- **Leads to better application responsiveness**
  - Interleave code download & execution
  - Faster startup times
  - Rarely executed code is rarely downloaded
Doloto: the Steps

1. **[training]** Runtime training to collect access profile

2. **[rewriting]** Function rewriting or “stubbing” for on-demand code loading

3. **[prefetch]** Background prefetch of clusters as the application is running
Runtime Picture

Server

Cluster c1
- \( f_1 \)
- \( f_2 \)

Cluster c2
- \( f_3 \)
- \( f_4 \)

access profile
...

Client

Stub for \( f_i \)

Real \( f_i \)

Retrieve body of \( f_i \) in cluster \( c_j \) from server

Background prefetch queue

\[ c_1 \rightarrow c_2 \rightarrow c_3 \]
Instrument every function

Record the first-execute timestamp

Look for gaps to find clusters
1. **runtime training** to collect access profile (e.g., using Fiddler plugin).

2. **function rewriting** or “stubbing” for on-demand code loading.

3. **background prefetch** of clusters as the application is running.
1. **[training]** Runtime training to collect access profiles

2. **[rewriting]** Function rewriting or “stubbing” for on-demand code loading

3. **[prefetch]** Background prefetch of clusters as the application is running
Code Rewriting

- Rewrite JavaScript code one file at a time
- Recombine clusters into individual files
Automated Function Splitting

```javascript
var g = 10;
function f1()
{
    var x = g + 1;
    ...
    ...
    ...
    ...
    return ...
}

var g = 10;
var real_f1;
function f1()
{
    if (!real_f1)
    {
        var code = load("f1");
        real_f1 = eval(code);
        f1 = real_f1;
    }
    return real_f1.apply(this, arguments);
}

eval($exp("f1"), "");  // 21 chars
```
Architecture of Doloto

1. [**training**] Runtime training to collect access profiles

2. [**rewriting**] Function rewriting or “stubbing” for on-demand code loading

3. [**prefetch**] Background prefetch of clusters as the application is running
Cluster Prefetching

```javascript
var xhr = new XMLHttpRequest();
function next_cluster()
{
    xhr.open("http://code.server.com/next",
    /* asynchronous AJAX call */ true);

    xhr.onreadystatechange = handle_cluster;
    xhr.send(null);
}

function handle_cluster()
{
    if (xhr.readyState !== 4) { return; }
    var code = xhr.responseText;
    if (code === "") return; // last cluster

    // split code into function bodies
    for (let [func_name, func_code] of code) {
        func[func_name] = func_code;
    }

    // go fetch the next cluster
    next_cluster();
}

// initial invocation of next_cluster
// after the document is done loading
document.attachEvent("onload", next_cluster);
```
### Application Benchmarks

<table>
<thead>
<tr>
<th>Application</th>
<th>Download Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chi game</td>
<td>104</td>
</tr>
<tr>
<td>Bunny Hunt</td>
<td>16</td>
</tr>
<tr>
<td>Live.com</td>
<td>1,436</td>
</tr>
<tr>
<td>Live Maps</td>
<td>1,909</td>
</tr>
<tr>
<td>Google Spreadsheets</td>
<td>499</td>
</tr>
</tbody>
</table>

**Diagram:**
- **Stage 1:** Fill the Squid cache
  - **DOLOTO rewriting proxy**
  - **Squid caching proxy**
- **Stage 2:** Serve from the cache
  - **WAN simulator**
  - **Browser**
Size Savings with Doloto

Experiments
Conclusion

- Doloto: effective profile-driven optimization
- Our approach is general: Silverlight
- Enables larger more complex distributed apps
- Dynamic code loading for distributed applications of the future
Contact us

Ben Livshits (livshits@microsoft.com)

Doloto MSR