

SuiteWorld

Consolidate Scripts for Improved Performance

Peter Ries

Consulting Technical Director, Oracle | NetSuite

ORACLE NETSUITE

Safe harbor statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.



Agenda

1. Performance problems with scripts
2. Consolidate scripts to solve one source of this problem
3. Demonstration
4. Other considerations



Thank you Nivz!



The solution presented today would not be possible with help from

Nivz Meremilla

Senior NetSuite technical consultant



Scripts can cause performance issues.

That doesn't have to be the end of the story.

Problem 1: Too many User Event scripts deployed to a record

For instance, we can deploy 30 User Event scripts to Sales Orders



This can lead to slow performance if we're not careful

Problem 2: Only 10 Client scripts can be deployed to any one record

We can exceed this limit for some records if we have many automation requirements

- Sales Order and Purchase Order are most common types

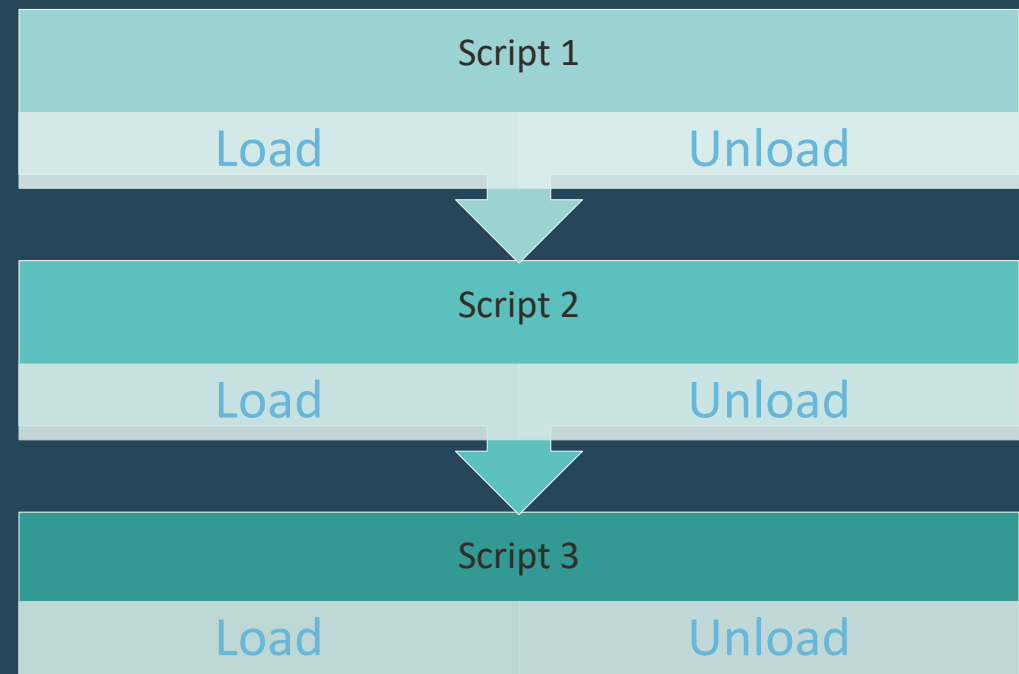
SuiteApps compound this problem, since we have no control over what they add to the account

We need a way to automate our client-side processes without exceeding this ten script limit

Every script we deploy adds overhead

For each script we deploy against a single record, NetSuite has to:

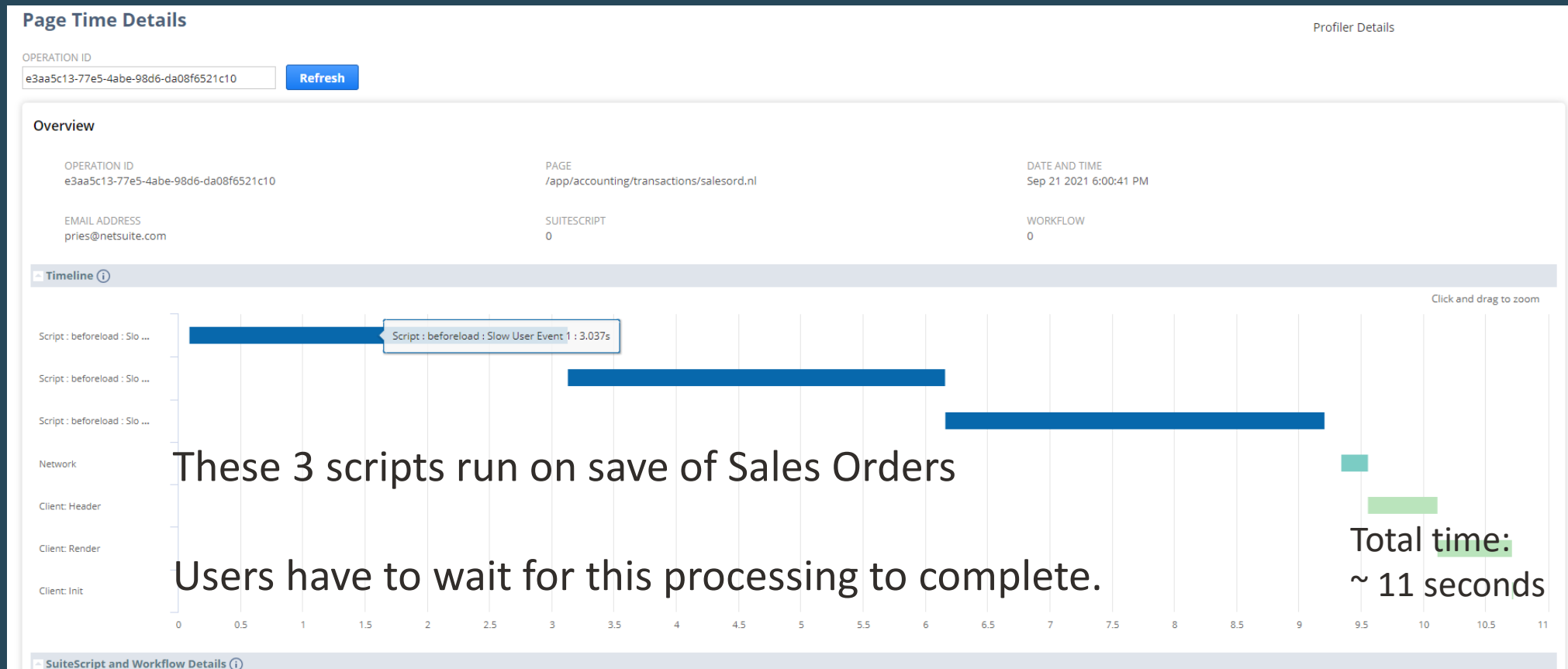
- Find and load the script object
- Process its code file
- Unload the script object



This adds time to the overall event, slowing down the system

Both problems can negatively impact processing efficiency

“The users are going to mutiny on us if we can’t fix this!”



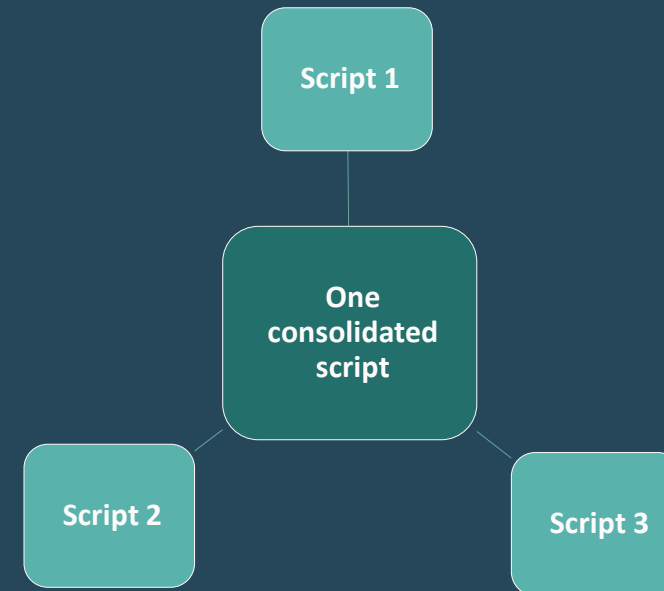
Suggestions for improvement

First:

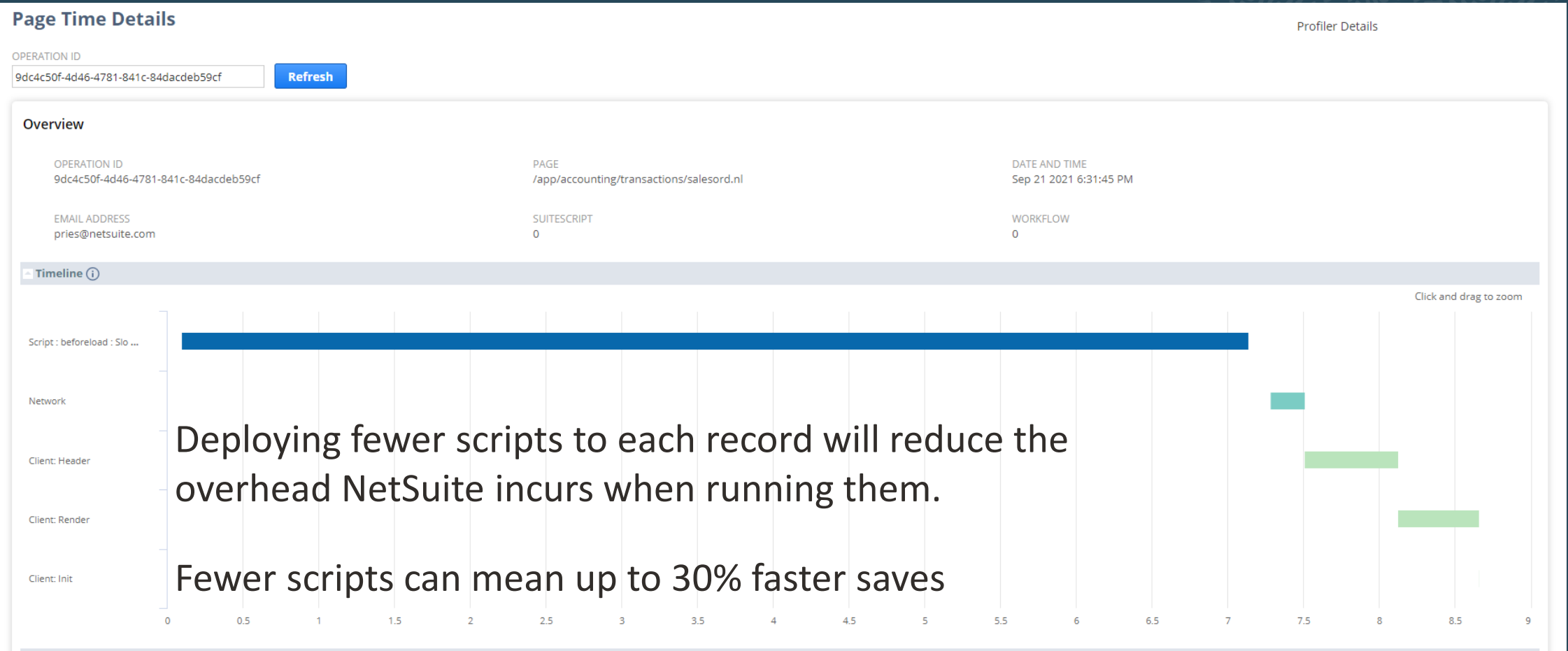
- Understand your current scripts, what they do.
- Remove any scripts which are no longer needed.
- Analyze your current scripts and optimize each script's performance as much as you can.
- The Scripted Record screen will help you with this.

Then:

- Consolidate your current scripts.
- Your goal is 1 entry point script per record.



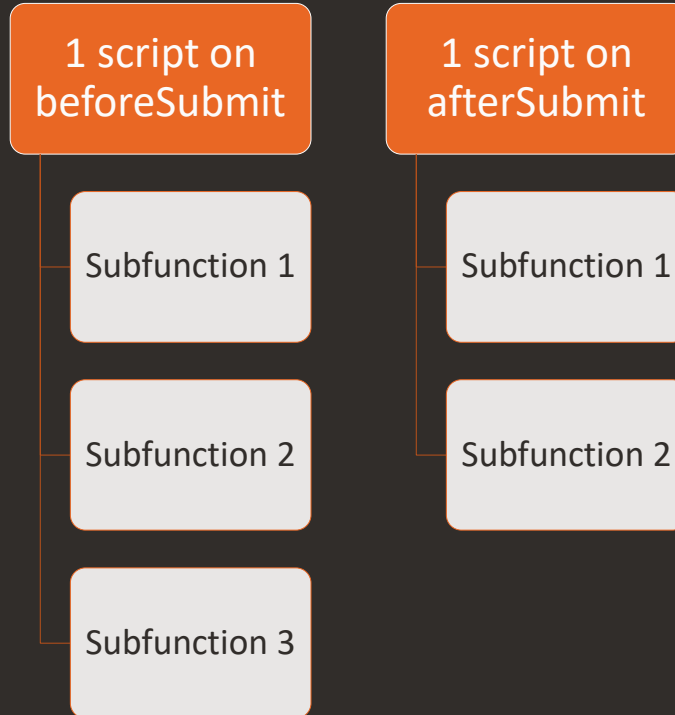
Consolidate your scripts!



How this works

For the scripts you have control over

Sales Order



Two approaches



Static code (faster to develop)



Hard-code the entry point script to call the subfunctions directly.

This is fast but harder to extend later.

Best option for Client scripts.

Dynamic (easier to extend over time)



Based on a custom record list, this script calls subfunctions for every record in the table.

With this, you can add more subfunctions without editing the entry point code.

The static approach – coding steps

1. Create a new code file – Sales Order Functions.js
2. Define a function for each event you want to automate. (e.g. afterSubmit and beforeLoad)
3. In the event's function, add hard-coded calls to the subfunctions (your existing scripts)

```
let afterSubmit = function(context) {  
    ProcessWarranties.afterSubmit(context, parameters);  
    ProcessSOValidations.afterSubmit(context);  
}
```

4. Edit the existing code files so they define all their work inside a namespace to avoid collisions.

```
let ProcessWarranties = {};  
ProcessWarranties.afterSubmit = function () { ... }
```

The dynamic approach – custom record type




1. Create a new custom record type – Consolidated Script
2. Create one record in that type named “User Event”
3. Create another new custom record type – Consolidated Script Subfunction
4. Define a set of Subfunction records, one per existing User Event script in your current set.

The dynamic approach – coding steps


1. Create a new code file – Consolidated_User_Event.js
2. Define a function for each event you want to automate. (e.g. afterSubmit and beforeLoad)
3. In the event's function, write a loop to get every Consolidated Script record / Subfunction record, and for each, use require() to load the related code file and execute its function.
4. Edit the existing code files so they define all their work inside a namespace to avoid collisions.

aftersubmit-processwarranties.js

```
let ProcessWarranties = {};  
ProcessWarranties.afterSubmit = function () { ... }
```

```
1. const beforeSubmit = (context) => {
2.     log.debug(stLogTitle, '**** START: Entry Point Invocation ****');
3.     try {
4.         const stRecordType = scriptContext.newRecord.type;
5.         processSubfunctions(context, recordType, 'User Event', 'beforeSubmit');
6.     }
7.     catch (e) {
8.         log.error('catch - beforeSubmit', JSON.stringify(e));
9.         throw e.message;
10.    }
11.    log.debug(stLogTitle, '**** END: Entry Point Invocation ****');
12.};
```



```
1. const processSubfunctions = (context, recordType, scriptType, event) => {
2.     const stLogTitle = 'processModules';
3.     let arrSubfunctions = searchSubfunctions(recordType, scriptType, event);
4.     for (let i = 0; i < arrModules.length; i++) {
5.         require([arrSubfunctions[i]], (module) => {
6.             module.beforeSubmit(context);
7.             log.debug('Remaining Units : '
8.                 + runtime.getCurrentScript().getRemainingUsage());
9.         });
10.    }
11.    };
```



Either approach – perform these steps in the NetSuite UI



-
1. Disable the existing Script objects in the account.
 2. Create and deploy the new entry point script against the record type.
 3. Test!

Demo



Let's see how this works in NetSuite. . .





Concerns and considerations

Potential problems

Be aware of the Usage / Governance limits since now we're running multiple subfunctions in one entry point script.

Control the order of execution.

Replace script parameters with a “settings” file or read them from a new custom record.

Maintenance over time

Teach your developers how to share access to code files with help from tools such as the SDF and your version control system (git, etc...).

Extend the approach by adding all new script functions to the consolidated script, instead of adding new script objects.

Enhancement / Ideas



With the Static approach, can you call functions once for each header field update on the record and then have one loop for the items list and call each function inside that loop?

Extend the same thinking to your Client scripts, except only the Static approach works well.

Consider applying this approach to all new script development, even on records which are not specifically impacted today. (Following this approach will avoid issues in the future!)

Other tips for improving your script performance

Apply as much logic on beforeSubmit instead of afterSubmit, wherever possible, and avoid client scripts when you can.

Never perform searches inside any type of loop in the code.

Use the fields on the Consolidate Script records to control the order in which the sub-functions execute.

See the Help topic named, “*Performance Best Practices*” for more helpful tips!

My new book – “Implementing NetSuite”

A new book being published by Packt Publishing later in 2021. It will be meant for anyone interested in learning how NetSuite can be implemented successfully and quickly.

Learn tips and tricks from a NetSuite professional.

Note: This is not an official product of Oracle | NetSuite.

For information about Peter’s new book, being published later in 2021, please visit:
ImplementingNetSuite.com



Thank you!



SuiteWorld



ORACLE NETSUITE

NEED EXPERT TRAINING AND SUPPORT?

Optimize with LCS & ACS

Join us in the Customer Success
Lounge in the Expo Hall!

- LCS – Training when you need it
- ACS – Proactive support

Slides from all presentations will be posted to SuiteAnswers



SuiteWorld