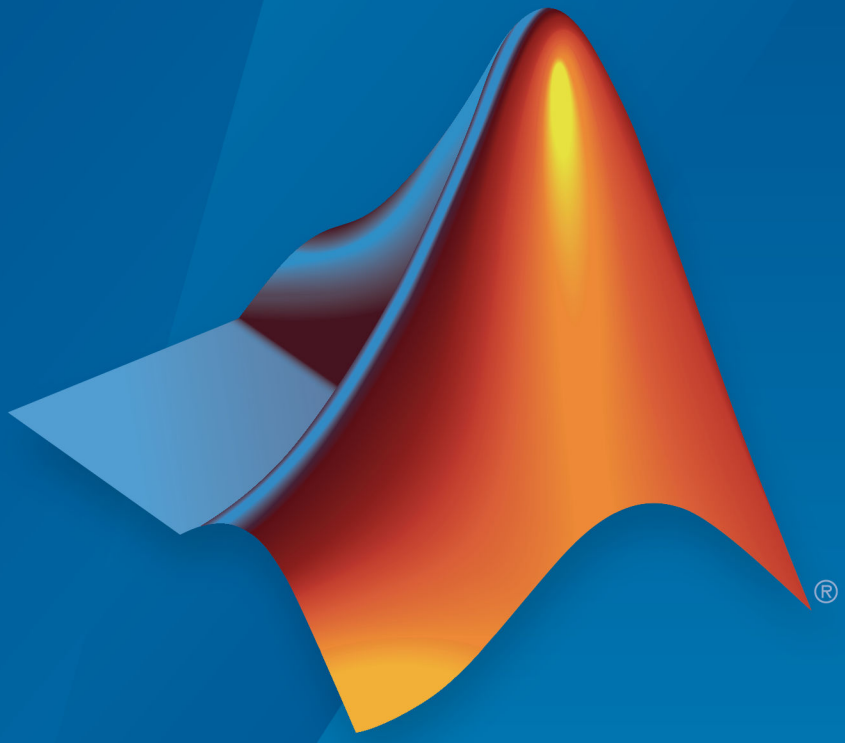
Polyspace® Bug Finder™ Server™

Getting Started Guide



R2019a

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

Sales and services: `www.mathworks.com/sales_and_services`

User community: `www.mathworks.com/matlabcentral`

Technical support: `www.mathworks.com/support/contact_us`

Phone: 508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

Polyspace® Bug Finder™ Server™ Getting Started Guide

Revision History

March 2019	Online Only	New for Version 3.0 (Release 2019a)

Contents

1 Install Polyspace Bug Finder Server

2 Run Polyspace Bug Finder Server After Code Submission

1

Install Polyspace Bug Finder Server

Polyspace Bug Finder Server Product Description

Identify software defects via static analyses running on server computers

Polyspace Bug Finder Server is a static analysis engine that identifies common classes of bugs in C and C++, including run-time errors, concurrency issues, and other coding defects. Polyspace Bug Finder Server also checks source code for adherence to coding rules (MISRA C, MISRA C++, JSF++), security rules (CWE, CERT-C, CERT-C++, ISO/IEC 17961), and custom rules.

With Polyspace Bug Finder Server you can monitor code metrics including cyclomatic complexity, stack usage, and HIS metrics at the project, file, and function levels. You can configure the server for use with various compilers, target processors, and RTOS environments, and automate execution with continuous integration systems using tools such as Jenkins. Code analysis results can be published to Polyspace Bug Finder Access™ for triage and resolution.

Support for industry standards is available through IEC Certification Kit IEC Certification Kit (for IEC 61508 and ISO 26262) and DO Qualification Kit (for DO-178).

Polyspace Products for Code Analysis and Verification

In this section...

Polyspace products use static analysis to check code for run-time errors, coding standard violations, security vulnerabilities and other issues. A static analysis tool such as Polyspace Code Prover™ can cover all possible execution paths through a program and track data flow along these paths following certain mathematical rules. The analysis can be much deeper than dynamic testing and expose potential run-time errors that might not be otherwise found in regular software testing. A static analysis tool such as Polyspace Bug Finder can scan a program quickly for more obvious run-time errors and coding constructs that potentially lead to run-time errors or unexpected results.

A software development process can use analysis results from Polyspace to complement dynamic testing. Using a product such as Code Prover, you can drastically reduce efforts in dynamic testing and focus only on areas where static analysis is unable to prove the absence of a run-time error. Using a product such as Bug Finder, you can maintain a list of potentially problematic coding practices and automatically check for these practices during development.

Using Polyspace Products in Software Development

The Polyspace suite of products supports all phases of a software development process:

- *Prior to code submission*:

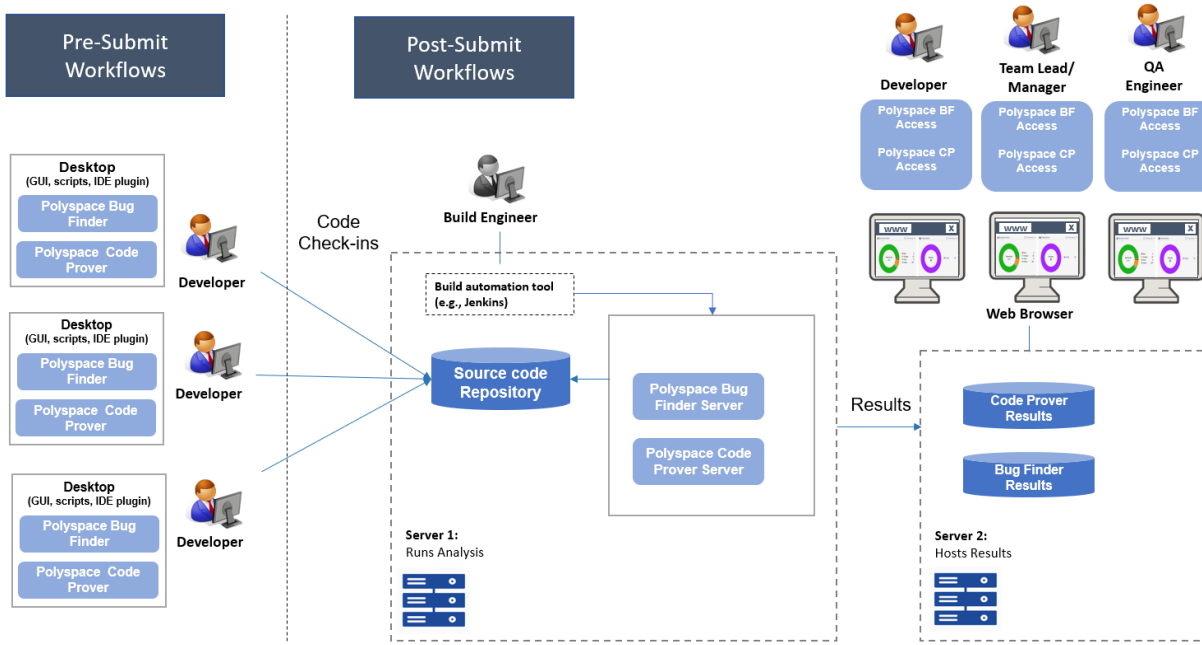
  Developers can run the Polyspace desktop products to check their code during development or right before submission to meet predefined quality goals.

  The desktop products can be integrated into IDEs such as Eclipse™ or run with scripts during compilation. The analysis results can be reviewed in IDEs such as Eclipse or in the graphical user interface of the desktop products.

- *After code submission*:

  The Polyspace server products can run automatically on newly committed code as a build step in a continuous integration process (using tools such as Jenkins). The analysis runs on a server and the results are uploaded to a web interface for collaborative review.

See “Polyspace Products for C/C++ Code” on page 1-4.



Note: Depending on the specifications, the same computer can serve as both Server 1 and Server 2.

When you use both the desktop and server products, your pre-submission workflow can transition smoothly to the post-submission workflow. See “Using Desktop and Server Products Together” on page 1-5.

Polyspace Products for C/C++ Code

Polyspace provides these products for desktop usage:

- **Polyspace Bug Finder** to check code for semantic errors that a compiler cannot detect (such as use of = instead of ==), concurrency issues, security vulnerabilities and other defects in C and C++ source code. The analysis can also detect some run-time errors.
- **Polyspace Code Prover** to perform a much deeper check and prove absence of overflow, divide-by-zero, out-of-bounds array access and other run-time errors in C and C++ source code.

Depending on your quality goals, you can run one or both products. See “Choose Between Polyspace Bug Finder and Polyspace Code Prover” on page 4-2.

Polyspace provides these products for server usage:

- **Polyspace Bug Finder Server** to run Bug Finder automatically on a server and upload the results to a web interface for review, and **Polyspace Bug Finder Access** to review the uploaded results.

  Typically, Polyspace Bug Finder Server runs on a few build servers and checks newly committed code as part of software build and testing. Each reviewer (developer, quality assurance engineer or development manager) has a Polyspace Bug Finder Access license to review the uploaded analysis results.
- **Polyspace Code Prover Server** to run Code Prover automatically on a server and upload the results to a web interface for review, and **Polyspace Code Prover Access** to review the uploaded results.

  Typically, Polyspace Code Prover Server runs on a few build servers and checks newly committed code as part of software build and testing. Each reviewer (developer, quality assurance engineer or development manager) has a Polyspace Code Prover Access license to review the uploaded analysis results.

Using Desktop and Server Products Together

In a software development workflow, you benefit most from using the desktop and server products together. Developers can run the desktop products while coding and fix or justify the issues found. At this stage, it is easy to rework the code because it is still under development.

After code submission, the server products can run a more comprehensive analysis. The analysis will reveal fewer issues if the developer has already fixed them before

submission. If the developer has triaged issues for fixing later or justified them, this information can be carried over to the server-side analysis so that fewer results need to be reviewed. The remaining results can be uploaded to the Polyspace Access web interface. Quality engineers can review these results and based on the severity of the results, assign them to developers for fixing.

The desktop and server products can be coordinated in these ways:

- You can use the same analysis configuration with both the desktop and server products. If you use the same analysis configuration, you see the same analysis results on the desktop and server side.

  At the same time, you can perform a more comprehensive checking on the server side. If you are running Bug Finder, you can increase the number of checkers for the server-side analysis compared to the desktop-side analysis. If you are running Code Prover, you can use stricter assumptions for the server-side analysis compared to the desktop-side analysis.

  The server side analysis can also run on more complete applications as opposed to the desktop side analysis, which runs on individual modules.
- If you enter comments on the desktop side to justify an analysis result, these comments can be reused on the server side. If you justify analysis results on the desktop side or set a status on them for fixing later, you are saved from repeating this work for the server-side analysis results.

  If you enter the comments in your source code as code annotations using a specific syntax, the server-side analysis can read the code annotations and import them to the server-side analysis results.
- You can configure your analysis on the desktop but run the analysis on a dedicated server. In this workflow:

  - You perform a one-time setup to enable communication between the desktop and server products using the distributed computing product, **MATLAB® Parallel Server™** .
  - During development, you trigger the analysis from a desktop product but the analysis runs on a server using a server product. The results are downloaded back to the desktop product for review.

  Since the analysis is offloaded to a server, this workflow saves processing power on the developer's desktop.

Polyspace Products for Ada Code

Polyspace provides these products for verifying Ada code:

- **Polyspace Client™ for Ada** to check Ada code for run-time errors on a desktop.
- **Polyspace Server for Ada** to check Ada code for run-time errors on a server.

You can either use the desktop product to run the analysis on your desktop, or a combination of the desktop and server products to run the analysis on a server. The analysis results are downloaded to your desktop for review.

If you have a Polyspace Code Prover Access license and have set up the web interface of Polyspace Code Prover Access, you can upload each individual Ada result from the Ada desktop products to the web interface for collaborative review.

See also:

- https://www.mathworks.com/products/polyspace-ada.html
- https://www.mathworks.com/products/polyspaceserverada/

See Also

Related Examples

- “Install Polyspace Desktop Products” (Polyspace Bug Finder)
- “Install Polyspace Server and Access Products” on page 1-8

Install Polyspace Server and Access Products

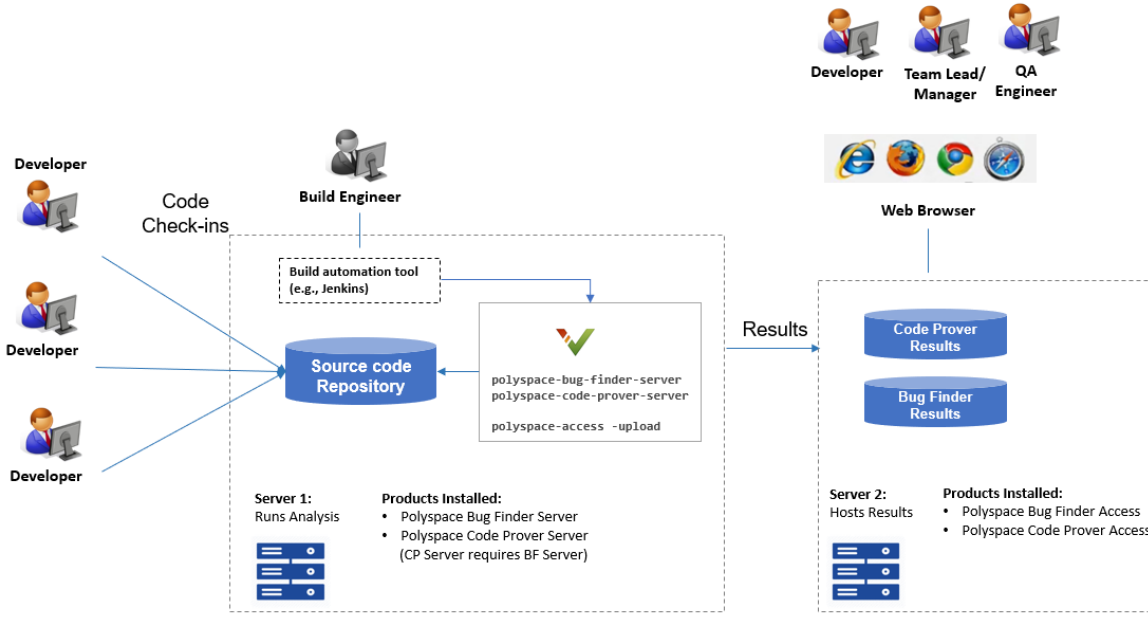
Polyspace checks C/C++ code for bugs, run-time errors, coding standard violations and other issues by using static analysis. You can run a Polyspace analysis on server class machines and review the analysis results on a web browser with these products:

- Polyspace Bug Finder Server for execution on servers and Polyspace Bug Finder Access for web browser based review of results
- Polyspace Code Prover Server for execution on servers and Polyspace Code Prover Access for web browser based review of results

By using these products with a build automation tool, you can incorporate a Polyspace analysis into continuous integration systems.

Workflow

In a continuous integration process, developers submit code to a shared repository. An automated build system builds and tests each submission at regular intervals or based on predefined triggers and integrates the code. You can run a Polyspace analysis as part of this process.



The workflow consists of three steps:

1. *Triggering Polyspace analysis*:

   When developers submit their code, a build automation tool such as Jenkins triggers the Polyspace analysis.

2. *Running Polyspace analysis*:

   The analysis runs using the products, Polyspace Bug Finder Server or Polyspace Code Prover Server. The results are uploaded to a server hosting the Polyspace Access web interface. The same computer can act as the server that runs the analysis and hosts the uploaded results.

3. *Reviewing analysis results*:

   To see the uploaded results, each reviewer must have a Polyspace Bug Finder Access and/or Polyspace Code Prover Access license to login to the Polyspace Access web interface.

Note that these steps describe a workflow after code submission. Prior to code submission, individual developers can:

- Run a Polyspace analysis and review the results on their desktops. See “Install Polyspace Desktop Products” (Polyspace Bug Finder).
- Offload a Polyspace analysis from their desktops to a remote server and review the downloaded results on their desktops. See “Install Products for Submitting Polyspace Analysis from Desktops to Remote Server” on page 1-14.

For an overview of all Polyspace products, see “Polyspace Products for Code Analysis and Verification” on page 1-3.

Product Installation

For this workflow, you must install the following on the computer(s) that act as the server.

Polyspace Products to Run Analysis

Install Polyspace Bug Finder Server and/or Polyspace Code Prover Server to run the analysis and upload results to a web interface for review.

Installation

Run the MathWorks® installer. Choose a license for Polyspace server products. You can get the installer and license by purchasing the product or requesting a trial. For detailed instructions, see “Installation, Licensing, and Activation”.

Note that you require Polyspace Bug Finder Server to install Polyspace Code Prover Server. For more efficient analysis, the computer that runs the analysis should have at least four physical cores, with at least 4 GB of memory per core. Beyond four cores, a Code Prover analysis cannot be parallelized further. However, a Bug Finder analysis can use the maximum number of available cores.

Polyspace Products to Review Results

Install the components required to host the web interface of Polyspace Access.

Each reviewer must have a Polyspace Bug Finder Access and/or Polyspace Code Prover Access license to view the results on this web interface.

- Each reviewer with a Polyspace Bug Finder Access license can review the dashboard with an overview of analysis results and Bug Finder result details.
- Each reviewer with a Polyspace Bug Finder Access and Polyspace Code Prover Access license can review the Code Prover result details (in addition to the dashboard and Bug Finder result details).

  Note that a Polyspace Code Prover Access license requires a Polyspace Bug Finder Access license.

Installation

See “Manage Polyspace Bug Finder Access Software” (Polyspace Bug Finder Access). The same steps apply to both Polyspace Bug Finder Access and Polyspace Code Prover Access.

Products to Schedule Polyspace Analysis

In a continuous integration workflow, you require a build automation tool to schedule the Polyspace analysis as part of a build process. The tool is not needed to run the Polyspace analysis or review the results but only to integrate the analysis into an overall software build and test workflow.

For instance, Jenkins is a popular open source tool that can be used to execute build scripts. For scripting convenience, a Polyspace plugin, available in Jenkins, defines a set of environment variables that can be used to point to the Polyspace server products.

Installation

Install a tool for continuous integration or build automation.

For instance, to install Jenkins, see the Jenkins website. In the Jenkins interface, select **Manage Jenkins** on the left. Select **Manage Plugin**. Search for the Polyspace plugin and then download and install the plugin.

Check Polyspace Installation

To check if the installation of Polyspace Bug Finder Server completed successfully:

1. Open a command window. Navigate to *polyspaceserverroot*\polyspace\bin (using cd). Here, *polyspaceserverroot* is the Polyspace Bug Finder Server installation folder, for instance, C:\Program Files\Polyspace Server\R2019a.
2. Enter the following:

```
polyspace-bug-finder-server -help
```

You should see the list of options allowed for a Bug Finder analysis. To check if the installation of Polyspace Code Prover Server completed successfully, you can perform the same steps for Polyspace Code Prover Server. Replace `polyspace-bug-finder-server` with `polyspace-code-prover-server`.

To check if the Polyspace Access web interface is set up for upload:

1. Navigate again to *polyspaceserverroot*\polyspace\bin.
2. Enter the following:

```
polyspace-access -host hostName -port portNumber -create-project testProject
```

   Here, *hostName* is the name of the server hosting the Polyspace Bug Finder Access web server. For a locally hosted server, use `localhost`. *portNumber* is the optional port number of the server. If you omit the port number, 9443 is used.

   If the connection is successful, a project called `testProject` should be created in the Polyspace Access web interface.
3. Open this URL in a web browser:

   ```
   https://hostName:portNumber/metrics/index.html
   ```

   Here, *hostName* and *portNumber* are the host name and port number from the previous step.

In the **PROJECT EXPLORER** pane on the Polyspace Access web interface, you should see the newly created project `testProject`.

Install Polyspace with Other MathWorks Products

If you install Polyspace with other MathWorks products such as MATLAB, you have to run the MathWorks installer twice.

- In the first run, choose the license that corresponds to the other MathWorks products, such as MATLAB, Simulink® or Embedded Coder®.
- In the second run, choose the license that corresponds to the Polyspace products.

In this workflow, products such as MATLAB and Simulink are installed in a different root folder from the Polyspace products. However, you can link the two installations and use MATLAB scripts to run Polyspace. See “Integrate Polyspace Server Products with MATLAB and Simulink”.

If you install both the Polyspace desktop and server products, you also have to run the installer twice with separate licenses. The desktop and server products are installed in separate root folders. For instance, in Windows®, the default root folders for an R2019a installation are:

- Polyspace desktop products: `C:\Program Files\Polyspace\R2019a`.

  This folder contains executables to run analysis with the products, Polyspace Bug Finder and/or Polyspace Code Prover.
- Polyspace server products: `C:\Program Files\Polyspace Server\R2019a`.

  This folder contains executables to run analysis with the products, Polyspace Bug Finder Server and/or Polyspace Code Prover Server.

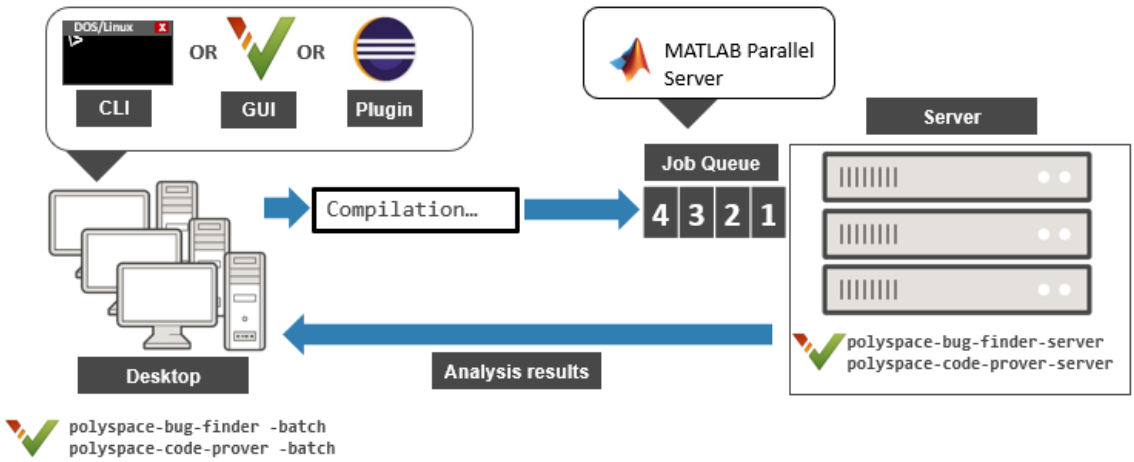
See Also

More About

- “Run Polyspace Bug Finder on Server and Upload Results to Web Interface” on page 2-2
- “Send E-mail Notifications with Polyspace Bug Finder Results” on page 2-10
- “Install Polyspace Desktop Products” (Polyspace Bug Finder)

Install Products for Submitting Polyspace Analysis from Desktops to Remote Server

You can perform a Polyspace analysis locally on your desktop or offload the analysis to one or more dedicated remote servers. This topic shows how to set up the dispatch of Polyspace analysis from desktop clients to remote servers. Once configured, you can send the Polyspace analysis to a remote server and view the downloaded results on your desktop.



Choose Between Local and Remote Analysis

To determine when to use local or remote analysis, use the rules listed in this table.

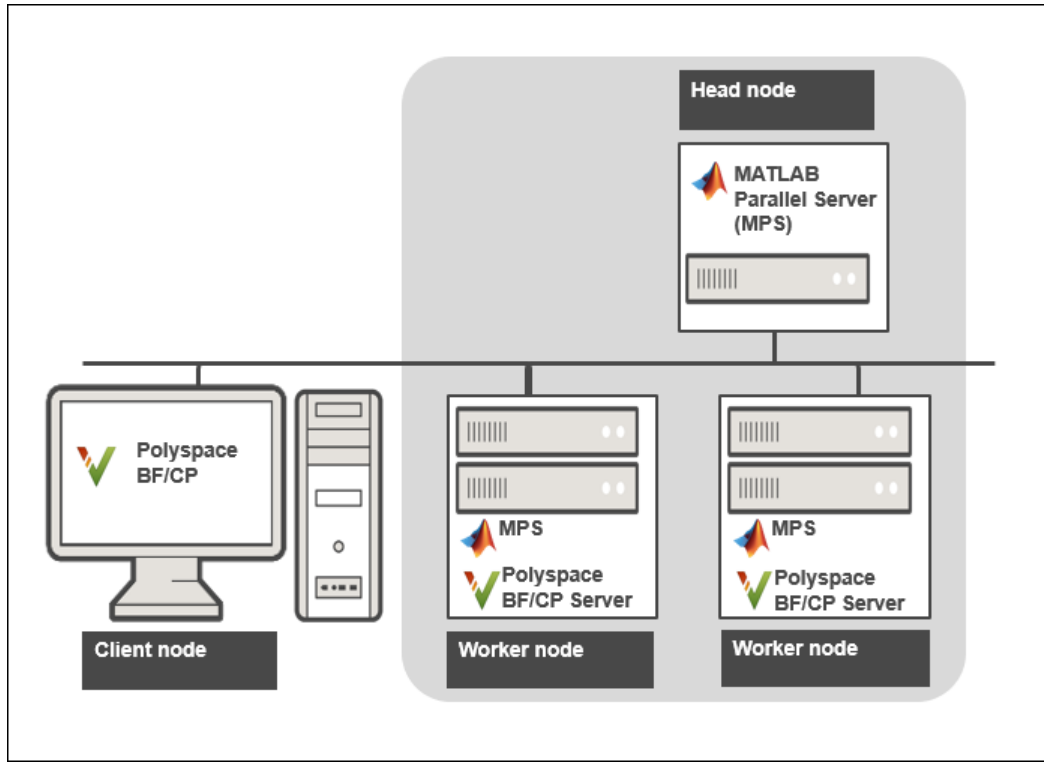
Type	When to Use
Remote	Source files are large and execution time of analysis is lengthy. Typically, a Code Prover analysis takes significantly longer than a Bug Finder analysis and benefits from running on a dedicated server.
Local	Source files are small and execution time of analysis is short.

Requirements for Remote Analysis

A typical distributed network for running remote analysis consists of these parts:

- **Client nodes**: On the client node, you configure your Polyspace project or scripts, and then submit a job that runs Polyspace.
- **Head node**: The head node distributes the submitted jobs to worker nodes.
- **Worker node(s)**: The Polyspace analysis runs on a worker node.

In this workflow, you install the product MATLAB Parallel Server to manage submissions from multiple clients. An analysis job is created for each submission and placed in a queue. As soon as a worker node is available, the next analysis job from the queue is run on the worker.



In the simplest remote analysis configuration, the same computer can serve as the head node and worker node. Note that you can run one Polyspace analysis on one worker only. You cannot distribute the analysis over multiple workers. Only if you submit more than one analysis job, you can distribute the jobs over multiple workers.

This table lists the product requirements for remote analysis.

Location	Requirements	Installation
Client node	Polyspace Bug Finder A Polyspace Bug Finder license is sufficient to trigger a Bug Finder or Code Prover analysis on the server, and review the downloaded analysis results.	Run the MathWorks installer on the client desktops. Choose a license for Polyspace desktop products. For detailed instructions, see “Installation, Licensing, and Activation”.
Head node	MATLAB Parallel Server (earlier called MATLAB Distributed Computing Server)	Run the MathWorks installer on the server(s). Choose a license for MATLAB Parallel Server installation. For detailed instructions, see “Integrate MATLAB Job Scheduler for Network License Manager” (MATLAB Parallel Server).
Worker nodes	• MATLAB Parallel Server (earlier called MATLAB Distributed Computing Server) • Polyspace Bug Finder Server • Polyspace Code Prover Server (if you choose to run Code Prover)	To install: • MATLAB Parallel Server, run the MathWorks installer on the server(s). Choose a license for MATLAB Parallel Server installation. • Polyspace Bug Finder Server and/or Polyspace Code Prover Server, run the MathWorks installer. Choose a license for Polyspace server products.

Configure and Start Server

On the computers that act as the worker nodes of the server, you install MATLAB Parallel Server and the Polyspace server products in two separate folders. The MATLAB Parallel Server installation must know where the Polyspace server products are located so that it can route the Polyspace analysis. To link the two installations, specify the paths to the root folder of the Polyspace server products in your MATLAB Parallel Server installations.

Then configure and start MATLAB Parallel Server (the `mjs` service) on all computers that act as the head node and worker nodes.

Configure mjs Service Settings

Before starting services, you must configure the `mjs` service settings.

1. Navigate to *`matlabroot`*`\toolbox\distcomp\bin`, where *`matlabroot`* is the MATLAB Parallel Server installation folder, for instance, `C:\Program Files\MATLAB\R2019a`.
2. Modify the file `mjs_def.bat` (Windows) or `mjs_def.sh` (Linux®). To edit and save the file, you have to open your editor in administrator mode.

   Read the instructions in the file and uncomment the lines as needed. At a minimum, you might have to uncomment these lines:

   - Hostname:

     ```
     REM set HOSTNAME=%strHostname%.%strDomain%
     ```

     in Windows or

     ```
     #HOSTNAME=`hostname -f`
     ```

     in Linux. Explicitly specify your computer host name.
   - Security level:

     ```
     REM set SECURITY_LEVEL=
     ```

     in Windows or

```
#SECURITY_LEVEL=""
```

in Linux. Explicitly specify a security level.

Otherwise, you might see an error later when starting the job scheduler.

Specify Polyspace Installation Paths

When you offload an analysis using a Polyspace desktop product installation, the server must run the analysis using a Polyspace server product installation from the same release. For instance, if you offload an analysis from an R2019a desktop product, the analysis must run using the R2019a server product. To ensure that the correct Polyspace server product is used, you must specify the installation paths of the Polyspace server products in your MATLAB Parallel Server installations.

If you use multiple releases of Polyspace desktop and server products, the MATLAB Parallel Server release must be the later one. For instance, if you offload analysis jobs using both R2019a and R2019b Polyspace desktop and server products, the MATLAB Parallel Server installation must be an R2019b one.

To specify the Polyspace installation paths, navigate to *`matlabroot`*`\toolbox\distcomp\bin\`. Here, *`matlabroot`* is the MATLAB installation folder, for instance, `C:\Program Files\MATLAB\R2019a`. Then, depending on the releases of the Polyspace products that run the analysis, use one of these methods:

- **Specify Paths to Polyspace Releases R2019a and Later**

  Uncomment and modify the following line in the file `mjs_polyspace.conf`. To edit and save the file, you have to open your editor in administrator mode.

  ```
  POLYSPACE_SERVER_R#####_ROOT=polyspaceserverroot
  ```

  Here, `R#####` is the release number, for instance, `R2019a`, and *`polyspaceserverroot`* is the installation path of the server products, for instance:

  ```
  C:\Program Files\Polyspace Server\R2019a
  ```

  To specify multiple releases, add a declaration for each release. For instance, the specification for an R2019a and R2019b Windows installation of the server products can be like this:

```
POLYSPACE_SERVER_R2019A_ROOT=C:\Program Files\Polyspace Server\R2019a
POLYSPACE_SERVER_R2019B_ROOT=C:\Program Files\Polyspace Server\R2019b
```

- **Specify Paths to Polyspace Releases Prior to R2019a**

  Prior to R2019a, you need to specify a Polyspace release only if you want MATLAB Parallel Server to handle submissions from clients with multiple releases of Polyspace. You need to specify only the earlier release of Polyspace. For instance, if an R2018b installation of MATLAB Parallel Server needs to handle analysis jobs from both an R2018b and R2018a installation, specify only the path to the R2018a installation.

  Edit the file `mjs_def.bat` or `mjs_def.sh` (located in *`matlabroot`*`\toolbox\distcomp\bin\`) to refer to the earlier release. Find the line with `MJS_ADDITIONAL_MATLABROOTS` and edit it as follows. To edit and save the file, you have to open your editor in administrator mode.

  ```
  set MJS_ADDITIONAL_MATLABROOTS=othermatlabroot
  ```

  Here, *`othermatlabroot`* is the installation path of the Polyspace products from the earlier release, for instance:

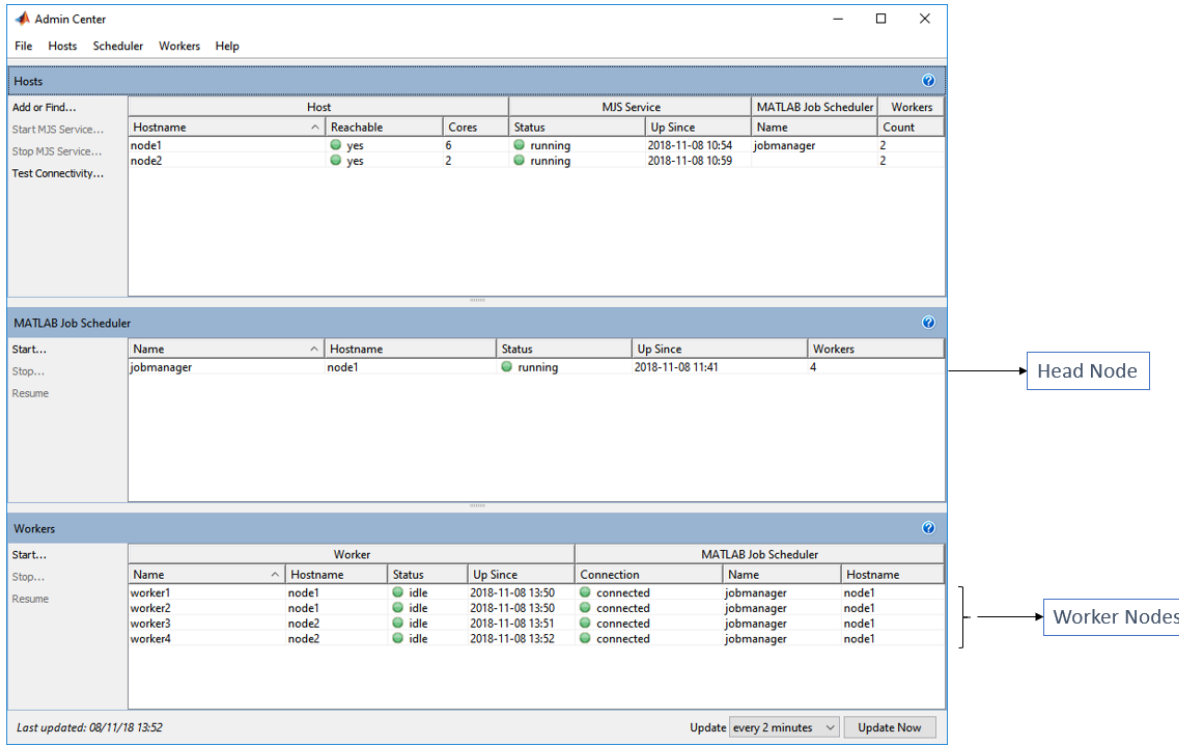
  ```
  C:\Program Files\MATLAB\R2018a
  ```

Start mjs Service and Assign as Head Node or Worker Node

To configure a server with multiple workers, start the service that runs a job scheduler (the `mjs` service) on the computer that acts as the head node and all computers that act as worker nodes. In the simplest configuration, the same computer can act as the head node and a worker node.

To set up a cluster with one head node and several workers, on the computer that acts as the head node:

1. Open the **Admin Center** window. Navigate to *`matlabroot`*`/toolbox/distcomp/bin` and execute the file `admincenter.bat` (Windows) or `admincenter.sh` (Linux). Here, *`matlabroot`* is the MATLAB installation folder, for instance, `C:\Program Files\MATLAB\R2019a`.



2. In the **Hosts** section, add the host names of all computers that you want to use as head and worker nodes of the cluster. Start the `mjs` service.

   The service uses the settings specified in the file `mjs_def.bat` (Windows) or `mjs_def.sh` (Linux).

3. Right-click each host. Select either **Start MJS** (head node) or **Start Workers** (worker nodes).

   The hosts appear in the **MATLAB Job Scheduler** or **Workers** section. In each section, select the host and click **Start** to start the MATLAB Job Scheduler or the workers.

Selecting a computer as host starts the `mjs` service on that computer. You must have permission to start services on other computers in the network. For instance, on Windows, you must be in the Administrators group for other computers where you want

to start the `mjs` service. Otherwise, you have to start the `mjs` services individually on each computer that acts as a worker.

For more details and command-line workflows, see:

- “Integrate MATLAB Job Scheduler for Network License Manager” (MATLAB Parallel Server)
- `mjs`

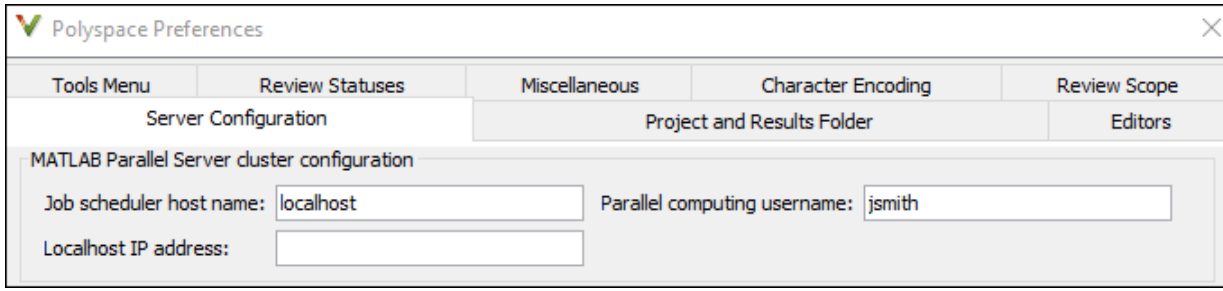
Configure Client

Configure the client node so that it can communicate with the computer that serves as the head node of the MATLAB Parallel Server cluster.

Configure the client node through the Polyspace environment preferences:

1. Select **Tools > Preferences**.
2. Click the **Server Configuration** tab. Under **MATLAB Parallel Server cluster configuration**:
   a. In the **Job scheduler host name** field, specify the computer for the head node of the cluster. This computer hosts the MATLAB job scheduler.

      If the port used on the computer hosting the MATLAB job scheduler is different from 27350, enter the port name explicitly with the notation *`hostName`*`:`*`portNumber`*.
   b. Due to the network setting, the job scheduler may be unable to connect back to your local computer. If so, enter the IP address of the client computer in the **Localhost IP address** field.



Offload Polyspace Analysis from Desktop to Server

Once the configuration is over, you can offload an analysis from a Polyspace desktop product installation to a remote server. You can do one of the following:

- Start a remote analysis from the user interface of the Polyspace desktop products.

  See “Send Polyspace Analysis from Desktop to Remote Servers”.
- Start a remote analysis with Windows or Linux scripts.

  See “Send Polyspace Analysis from Desktop to Remote Servers Using Scripts”. In the simplest configuration, the same computer can be used as a client and server. For a simple tutorial that uses this configuration and walks through all the steps for offloading a Polyspace analysis, see “Send Bug Finder Analysis from Desktop to Locally Hosted Server” on page 3-2.
- Start a remote analysis with MATLAB scripts.

  See “Run Analysis on Server” (Polyspace Bug Finder).

See Also

Related Examples

- “Send Polyspace Analysis from Desktop to Remote Servers”

- “Send Polyspace Analysis from Desktop to Remote Servers Using Scripts”
- “Send Bug Finder Analysis from Desktop to Locally Hosted Server” on page 3-2
- “Integrate MATLAB with Third-Party Schedulers” (MATLAB Parallel Server)
- “Troubleshoot Common Problems” (MATLAB Parallel Server)

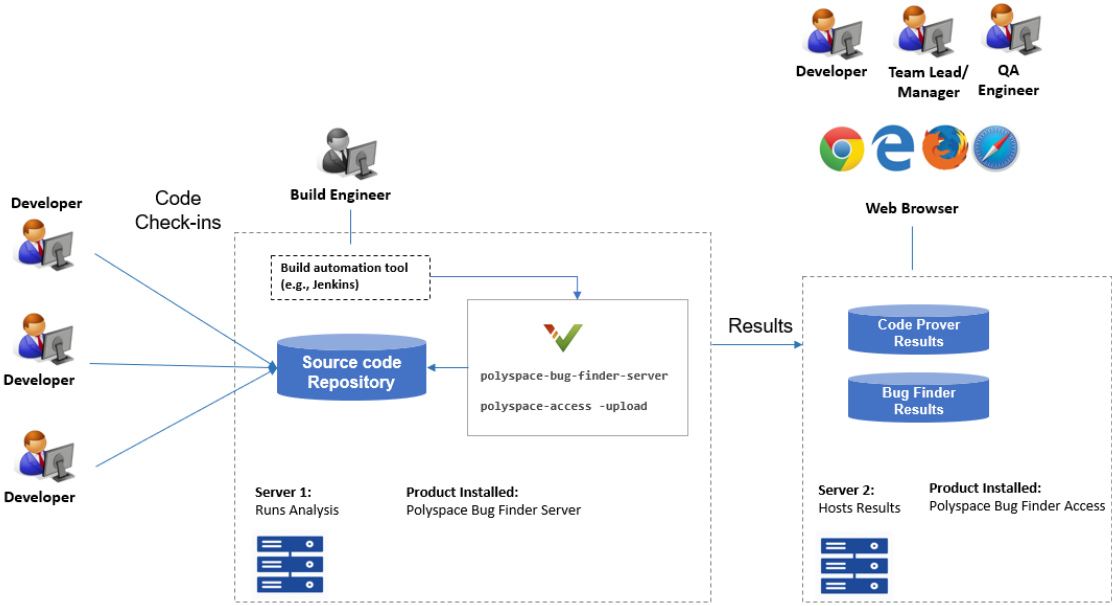
Run Polyspace Bug Finder Server After Code Submission

Run Polyspace Bug Finder on Server and Upload Results to Web Interface

Polyspace Bug Finder Server checks C/C++ code for defects and coding standard violations and uploads findings to a web interface for code review.

You can run Bug Finder as part of continuous integration. Set up scripts that run a Bug Finder analysis at regular intervals or based on new submissions. The scripts can upload the analysis results for review in the Polyspace Access web interface and optionally send emails to owners of source files with Polyspace findings. The owners can open the web interface with only the new findings from their submission, and fix or justify the issues.

In a typical project or team, Polyspace Bug Finder Server runs periodically on a few testing servers and uploads the results for review. Each developer and quality engineer in the team has a Polyspace Bug Finder Access license to view the results in the web interface for further investigation and bug fixing.



Note: Depending on the specifications, the same computer can serve as both Server 1 and Server 2.

Prerequisites

To run a Bug Finder analysis on a server and review the results in the Polyspace Access web interface, you must perform a one-time setup.

- To run the analysis, you must install one instance of the Polyspace Bug Finder Server product.
- To upload results, you must set up the components required to host the web interface of Polyspace Access.
- To view the uploaded results, you (and each developer reviewing the results) must have one Polyspace Bug Finder Access license.

See “Install Polyspace Server and Access Products” on page 1-8.

Check Polyspace Installation

To check if Polyspace Bug Finder Server is installed:

1. Open a command window. Navigate to *`polyspaceserverroot`*`\polyspace\bin` (using `cd`). Here, *`polyspaceserverroot`* is the Polyspace Bug Finder Server installation folder, for instance, `C:\Program Files\Polyspace Server\R2019a`.
2. Enter the following:

   ```
   polyspace-bug-finder-server -help
   ```

You should see the list of options allowed for a Bug Finder analysis.

To check if the Polyspace Access web interface is set up for upload:

1. Navigate again to *`polyspaceserverroot`*`\polyspace\bin`.
2. Enter the following:

   ```
   polyspace-access -host hostName -port portNumber -create-project testProject
   ```

   Here, *`hostName`* is the name of the server hosting the Polyspace Bug Finder Access web server. For a locally hosted server, use `localhost`. *`portNumber`* is the optional port number of the server. If you omit the port number, `9443` is used.

If the connection is successful, a project called `testProject` should be created in the Polyspace Access web interface.

You are prompted for your login and password each time you use the `polyspace-access` command. To avoid entering login information each time, you can provide the login and an encrypted version of your password with the command. To create an encrypted password, enter:

```
polyspace-access -encrypt-password
```

Enter your login and password. Copy the encrypted password and provide this encrypted password with the `-encrypted-password` option when using the `polyspace-access` command.

3. Open this URL in a web browser:

   `https://hostName:portNumber/metrics/index.html`

   Here, *`hostName`* and *`portNumber`* are the host name and port number from the previous step.

In the **PROJECT EXPLORER** pane on the Polyspace Access web interface, you should see the newly created project `testProject`.

Run Bug Finder on Sample Files

To run Bug Finder, open a command window in your operating system.

1. Use the `polyspace-bug-finder-server` command to run a Bug Finder analysis.
2. Use the `polyspace-access` command to upload the results to the Polyspace Access web interface.

To save typing the full path to the command, add the path *`polyspaceserverroot`*`\polyspace\bin` to the `Path` environment variable on your operating system.

Try out the commands on sample files provided with your Polyspace installation.

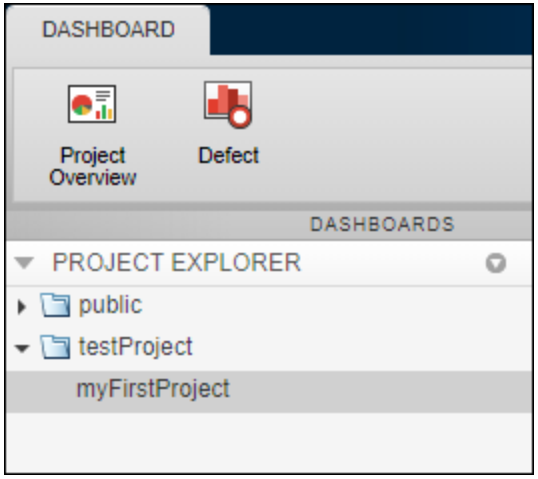
1. Copy the sample source files from *`polyspaceserverroot`*`\polyspace\examples\cxx\Bug_Finder_Example\sources` to another folder where you have write permissions. Navigate to this folder (using `cd`).

2 Enter the following:

```
polyspace-bug-finder-server -sources numerical.c,dataflow.c -I . -checkers nume
polyspace-access -host hostName -port portNumber
                 -login username -encrypted-password pwd
                 -create-project testProject
polyspace-access -host hostName -port portNumber
                 -login username -encrypted-password pwd
                 -upload . -project myFirstProject -parent-project testProject
```

Here, *username* is your login name and *pwd* is the encrypted password that you created previously. See “Check Polyspace Installation” on page 2-3.

Refresh the Polyspace Access web interface. You should see a folder `testProject` on the **PROJECT EXPLORER** pane. The folder contains one project `myFirstProject`.



To see the results in the project, click **Review**. For more information, see Polyspace Bug Finder Access documentation. You can also access the documentation using the  button in the upper right of the Polyspace Access interface.

For the full list of options available for a Bug Finder analysis, see “Analysis Options”. To open the Bug Finder documentation in a help browser, enter:

```
polyspace-bug-finder-server -doc
```

Sample Scripts for Bug Finder Analysis on Servers

Instead of typing the commands at the command line, you can write scripts to run the analysis. The scripts can execute each time you add or modify source files.

A sample Windows batch file with the previous commands is shown below. Here, the path to the Polyspace installation is specified in the script. To use this script, replace `polyspaceserverroot` with the path to your installation. You must have generated the encrypted password for use in the scripts. See “Check Polyspace Installation” on page 2-3.

```
echo off
set POLYSPACE_PATH=polyspaceserverroot\polyspace\bin
set LOGIN=-host hostName -port portNumber -login username -encrypted-password pwd
"%POLYSPACE_PATH%\polyspace-bug-finder-server" -sources numerical.c,dataflow.c -I .^
 -checkers numerical,data_flow -results-dir .
"%POLYSPACE_PATH%\polyspace-access" %LOGIN% -create-project testProject
"%POLYSPACE_PATH%\polyspace-access" %LOGIN% -upload . -project myFirstProject
              -parent-project testProject
pause
```

A sample Linux shell script with the previous commands is shown below.

```
POLYSPACE_PATH=polyspaceserverroot/polyspace/bin
LOGIN=-host hostName -port portNumber -login username -encrypted-password pwd
${POLYSPACE_PATH}/polyspace-bug-finder-server -sources numerical.c,dataflow.c -I .\
 -checkers numberical,data_flow -results-dir .
${POLYSPACE_PATH}/polyspace-access $LOGIN -create-project testProject
${POLYSPACE_PATH}/polyspace-access $LOGIN -upload . -project myFirstProject
     -parent-project testProject
```

Specify Sources and Options in Separate Files from Launching Scripts

Instead of listing the source files and analysis options within the launching scripts, you can list them in separate text files.

- You specify the text file listing the sources by using the option `-sources-list-file`.
- You specify the text file listing the analysis options by using the option `-options-file`.

By removing the source files and analysis option specifications from the launching scripts, you can modify these specifications as required with new code submissions while keeping the launching script untouched.

In the preceding example, you can modify the `polyspace-bug-finder-server` command to use these text files. Instead of:

```
polyspace-bug-finder-server -sources numerical.c,dataflow.c
                     -I . -checkers numerical,data_flow -results-dir .
```

Use:

```
polyspace-bug-finder-server -sources numerical.c,dataflow.c
              -I . -checkers numerical,data_flow -results-dir .
```

Here:

- `sources.txt` lists the source files in separate lines:

  ```
  numerical.c
  dataflow.c
  ```

- `polyspace_opts.txt` lists the analysis options in separate lines:

  ```
  -I .
  -checkers numerical,data_flow
  ```

Typically, your source files are specified in a build command (makefile). Instead of specifying the source files directly, you can trace the build command to create a list of source specifications. See `polyspace-configure`.

Complete Workflow

In a typical continuous integration workflow, you run a script that executes these steps:

1. Extract Polyspace options from your build command.

   For instance, if you use makefiles to build your source code, you can extract analysis options from the makefile.

   ```
   polyspace-configure -output-options-file compile_opts make
   ```

   See also:

   - `polyspace-configure`
   - “Create Polyspace Analysis Configuration from Build Command”

2. Run the analysis with the previously created options file. Append a second options file with the remaining options required for the analysis.

   ```
   polyspace-bug-finder-server -options-file compile_opts -options-file run_opts
   ```

   See “Prepare Scripts for Polyspace Analysis”.

3. Upload the results to Polyspace Bug Finder Access.

   ```
   polyspace-access login -upload resultsFolder -project projName
         -parent-project parentProjName
   ```

   Here, *`login`* is the combination of options required to communicate with the web server hosting Polyspace Bug Finder Access:

   ```
   -host hostName -port portNumber -login username -encrypted-password pwd
   ```

   `resultsFolder` is the folder containing the Polyspace results. *`projName`* and *`parentProjName`* are names of the project and parent folder as they would appear in the Polyspace Access web interface.

4. Optionally, send e-mail notifications to developers with new results from their submission. The email contains attachments with links to the results in the Polyspace Access web interface.

   See “Send E-mail Notifications with Polyspace Bug Finder Results” on page 2-10.

See examples of scripts executing these steps in “Sample Scripts for Polyspace Analysis with Jenkins”.

See Also

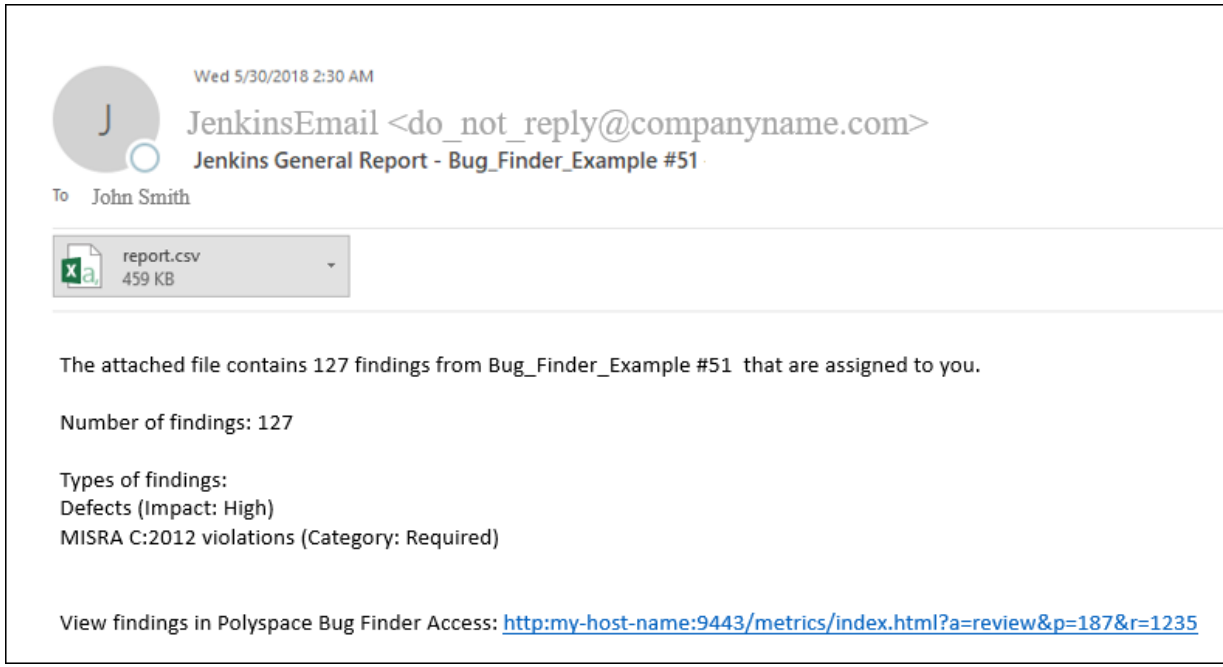
`polyspace-access` | `polyspace-bug-finder-server`

More About

- “Send E-mail Notifications with Polyspace Bug Finder Results” on page 2-10
- “Send Bug Finder Analysis from Desktop to Locally Hosted Server” on page 3-2
- “Analysis Options”

Send E-mail Notifications with Polyspace Bug Finder Results

If you run a Polyspace analysis as part of continuous integration, each new code submission produces new results. You not only see new results in components that were modified but also in components that depended on the modified components. You can set up e-mail alerts so that component owners get notified when new Polyspace results appear in their components.



Creating E-mail Notifications

To create e-mail notifications:

1. Export new analysis results to a tab-delimited text file (`.tsv` format). For each result, the file contains links to open the result in the Polyspace Access web interface.

   Apply filters to export specific types of results, for instance, defects with high impact. If required, you can also apply additional filters to the exported files using search and replace utilities. See “Export Results for E-mail Attachments” on page 2-13.
2. Send an email with the results file in attachment.

   For instance, if you use an e-mail plugin in Jenkins, you can create a post-build step to send an e-mail after the analysis is complete.

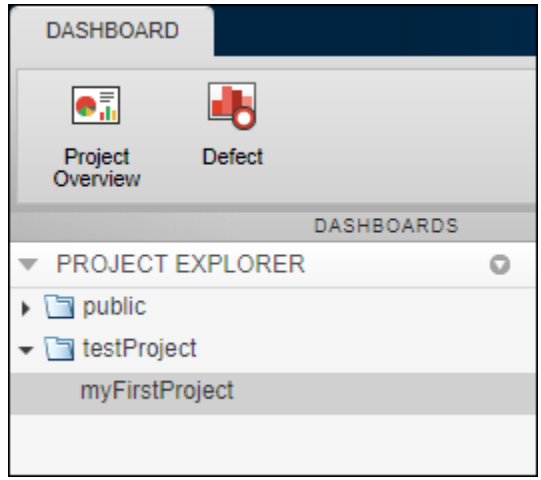
If you use the Polyspace plugin in Jenkins, you can use Polyspace helper utilities for the entire e-mail notification process. See “Sample Scripts for Polyspace Analysis with Jenkins”.

Alternatively, results can be directly assigned to owners based on their file paths. You can set up email notifications that exports a separate results file per owner and sends an email to each owner with the corresponding results file in attachment. See “Assign Owners and Export Assigned Results” on page 2-13.

Prerequisites

To run this tutorial:

- You must have uploaded some result in the Polyspace Bug Finder Access interface. If you complete the tutorial “Run Polyspace Bug Finder on Server and Upload Results to Web Interface” on page 2-2, you should see a folder `testProject` on the **PROJECT EXPLORER** pane. The folder contains one project `myFirstProject`.



To see the results in the project, with `myFirstProject` selected, click the **Review** button. You see a list of defects. The **Information** column shows the impact of the defects. In this tutorial, only high-impact defects will be exported for e-mail attachments.

- You must be able to interact with the Polyspace Bug Finder Access interface from the command line. For instance, navigate to *`polyspaceserverroot`*`\polyspace\bin` and enter:

  ```
  polyspace-access login -list-project
  ```

  Here. *`polyspaceserverroot`* is the Polyspace Bug Finder Server installation folder, for instance, `C:\Program Files\Polyspace Server\R2019a`.The variable *`login`* refers to the following combination of options. You provide these options with every use of the `polyspace-access` command.

  ```
  -host hostName -port portNumber -login username -encrypted-password pwd
  ```

  Here, *`hostName`* is the name of the Polyspace Bug Finder Access web server. For a locally hosted server, use `localhost`. *`portNumber`* is the optional port number of the server. If you omit the port number, `9443` is used. *`username`* and *`pwd`* refer to the login and an encrypted version of your password. To create an encrypted password, enter:

```
polyspace-access -encrypt-password
```

Copy the encrypted password and provide this password with later uses of the `polyspace-access` command.

Export Results for E-mail Attachments

You can export all results in a project or only certain types of results.

Open a command window. Navigate to the folder where you want to export the results.

- To export all results, enter the following:

```
polyspace-access login -export testProject/myFirstProject -output .\result.txt
```

- To export only defects with high impact, enter the following:

```
polyspace-access login -export testProject/myFirstProject -defects High
     -output .\result_high_impact.txt
```

Open each text file in a spreadsheet viewing utility such as Microsoft® Excel®. In the first file, you see all defects but in the second file, you only see the defects with high impact. To see only new defects compared to the previous analysis of the same project, use the option `-new-findings`.

You can configure your e-mail utility to send these exported files in attachment.

If required, you can also apply additional filters to the exported files using search and replace utilities. For instance, use search and replace utilities on the results file to include results only from specific files and functions. In Linux, you can use `grep` and `sed` to retain only results in specific files.

Assign Owners and Export Assigned Results

You can assign owners to results in specific files or folders. You can then export one result file per owner and send an email to each owner with the corresponding file in attachment.

You can assign owners in the Polyspace Access web interface or at the command line.

In this tutorial, assign all results in the file `numerical.c` to `jsmith` and all results in the file `dataflow.c` to `jboyd`.

```
polyspace-access login
     -set-unassigned-findings testProject/myFirstProject
     -owner jsmith -source-contains numerical.c
polyspace-access login
     -set-unassigned-findings testProject/myFirstProject
     -owner jboyd -source-contains dataflow.c
```

After assignment, export one results file per owner.

```
polyspace-access login
     -export testProject/myFirstProject -output .\results.txt -output-per-owner
```

These files contain the exported results:

- `results.txt` contains all results.
- `results_jsmith.txt` and `results_jboyd.txt` contains results assigned to `jsmith` and `jboyd` respectively.
- `results.txt.owners.list` contains the list of owners, in this case:

  ```
  jsmith
  jboyd
  ```

Before assigning owners to results, use the option `-dryrun` to perform a dry run of the assignments. Without performing the assignment, the option shows the files with results that are assigned and the owner that the results are assigned to.

See Also

`polyspace-access`

3

Run Polyspace Bug Finder Server Prior to Code Submission

Send Bug Finder Analysis from Desktop to Locally Hosted Server

You can perform a Polyspace analysis locally on your desktop or offload the analysis to one or more dedicated servers. This topic shows a simple server-client configuration for offloading the Polyspace analysis. In this configuration, the same computer acts as a client that submits a Polyspace analysis and a server that runs the analysis.

You can extend this tutorial to more complex configurations. For full setup and workflow instructions, see related links below.

Client-Server Workflow for Running Analysis

After the initial setup, you can submit a Polyspace analysis from a client desktop to a server. The client-server workflow happens in three steps. All three steps can be performed on the same computer or three different computers. This tutorial uses the same computer for the entire workflow.

1. **Client node**: You specify Polyspace analysis options and start the analysis on the client desktop. The initial phase of analysis upto compilation runs on the desktop. After compilation, the analysis job is submitted to the server.

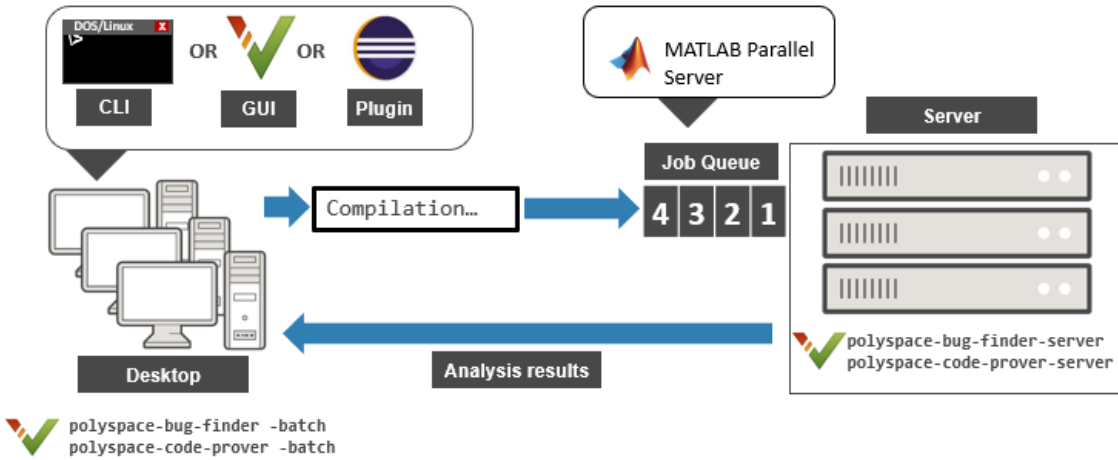
   You require the Polyspace desktop product, Polyspace Bug Finder on the computer that acts as the client node.

2. **Head node**: The server consists of a head node and several worker nodes. The head node uses a job scheduler to manage submissions from multiple client desktops. The jobs are then distributed to the worker nodes as they become available.

   You require the product MATLAB Parallel Server on the computer that acts as the head node.

3. **Worker nodes**: When a worker becomes available, the job scheduler assigns the analysis to the worker. The Polyspace analysis runs on the worker and the results are downloaded back to the client desktop for review.

   You require the product MATLAB Parallel Server on the computers that act as worker nodes. You also require the Polyspace server product, Polyspace Bug Finder Server to run the analysis.



See also “Install Products for Submitting Polyspace Analysis from Desktops to Remote Server” on page 1-14.

Prerequisites

This tutorial uses the same computer as client and server. You must install the following on the computer:

- Client-side product: Polyspace Bug Finder
- Server-side products: MATLAB Parallel Server and Polyspace Bug Finder Server

For more information, see “Install Products for Submitting Polyspace Analysis from Desktops to Remote Server” on page 1-14.

You must know the host name of your computer. For instance, in Windows, open a command-line terminal and enter:

```
hostname
```

Configure and Start Server

Stop Previous Services

If you started the services of MATLAB Parallel Server (`mjs` services) previously, make sure that you have stopped all services. In particular, you might have to:

- Check your temporary folder, for instance, `C:\Windows\Temp` in Windows, and remove the `MDCE` folder if it exists.
- Stop all services explicitly.

  Open a command-line terminal. Navigate to *`matlabroot`*`\toolbox\distcomp\bin` (using `cd`) and enter the following:

  ```
  mjs uninstall -clean
  ```

  Here, *`matlabroot`* is the MATLAB Parallel Server installation folder, for instance, `C:\Program Files\MATLAB\R2019a`.

If this is the first time you are starting the services, you do not have to do these steps.

Configure mjs Service Settings

Before starting services, you have to configure the `mjs` service settings. Navigate to *`matlabroot`*`\toolbox\distcomp\bin`, where *`matlabroot`* is the MATLAB Parallel Server installation folder, for instance, `C:\Program Files\MATLAB\R2019a`. Modify these two files. To edit and save these files, you have to open your editor in administrator mode.

- `mjs_def.bat` (Windows) or `mjs_def.sh` (Linux)

Read the instructions in the file and uncomment the lines as needed. At a minimum, you might have to uncomment these lines:

- Hostname:

  ```
  REM set HOSTNAME=myHostName
  ```

  in Windows or

  ```
  #HOSTNAME=`hostname -f`
  ```

  in Linux. Remove the `REM` or `#` and explicitly specify your computer host name.
- Security level:

  ```
  REM set SECURITY_LEVEL=
  ```

  in Windows or

  ```
  #SECURITY_LEVEL=""
  ```

  in Linux. Remove the `REM` or `#` and explicitly specify a security level.

Otherwise, you might see an error later when starting the job scheduler.

- `mjs_polyspace.conf`

  Modify and uncomment the line that refers to a Polyspace server product root. The line should refer to the release number and root folder of your Polyspace server product installation. For instance, if the R2019a release of Polyspace Bug Finder Server is installed in the root folder `C:\Program Files\Polyspace Server \R2019a`, modify the line to:

  ```
  POLYSPACE_SERVER_R2019A_ROOT=C:\Program Files\Polyspace Server\R2019a
  ```

  Otherwise, the MATLAB Parallel Server installation cannot locate the Polyspace Bug Finder Server installation to run the analysis.

Start Services

Start the `mjs` services and assign the current computer as both the head node and a worker node.

Navigate to *matlabroot*\toolbox\distcomp\bin, where *matlabroot* is the MATLAB Parallel Server installation folder, for instance, C:\Program Files\MATLAB\R2019a. Run these commands (directly at the command line or using scripts):

```
mjs install
mjs start
startjobmanager -name JobScheduler -remotehost hostname -v
startworker -jobmanagerhost hostname -jobmanager JobScheduler
    -remotehost hostname -v
```

Here, *hostname* is the host name of your computer. This is the host name that you specified in the file mjs_def.bat (Windows) or mjs_def.sh (Linux).

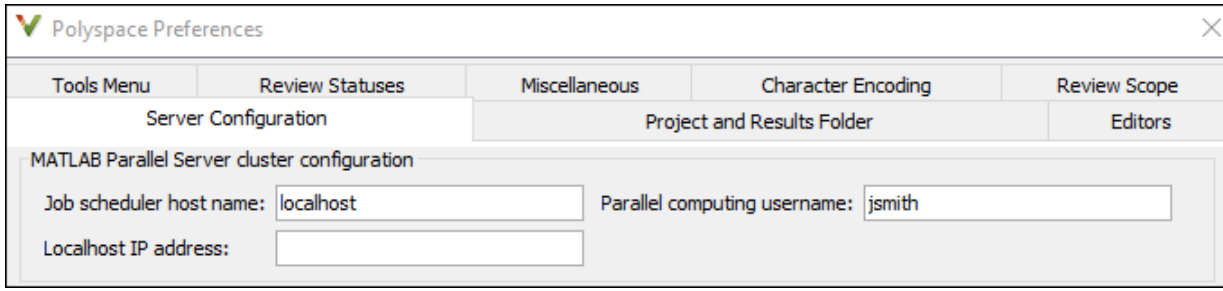
Instead of the command line, you can also start the services from the Admin Center interface. See “Install Products for Submitting Polyspace Analysis from Desktops to Remote Server” on page 1-14.

For more information on the commands, see “Configure Advanced Options for MATLAB Job Scheduler Integration” (MATLAB Parallel Server).

Configure Client

Open the user interface of the desktop product, Polyspace Bug Finder. Navigate to *polyspaceroot*\polyspace\bin, where *polyspaceroot* is the Polyspace desktop product installation folder, for instance, C:\Program Files\Polyspace\R2019a and double-click the polyspace executable.

Select **Tools** > **Preferences**. On the **Server configuration** tab, enter the host name of your computer for **Job scheduler host name**.



You are now set up for the server-client workflow.

Send Analysis from Client to Server

Run Bug Finder on the file `numerical.c` provided with your installation.

Before running these steps, to avoid entering full paths to the Polyspace executables, add the path *`polyspaceroot`*`\polyspace\bin` to the `PATH` environment variable on your operating system. Here *`polyspaceroot`* is the Polyspace desktop product installation folder, for instance, `C:\Program Files\Polyspace\R2019a`. To check if the path is already added, open a command line terminal and enter:

```
polyspace-bug-finder -h
```

If the path to the command is already added, you see the full list of options.

1. Copy the file `numerical.c` from *`polyspaceroot`*`\polyspace\examples\cxx\Bug_Finder_Example\sources` to a folder with write permissions.
2. Open a command terminal. Navigate to the folder where you saved `numerical.c` and enter the following:

   ```
   polyspace-bug-finder -sources numerical.c -checkers numerical
       -results-dir . -batch -scheduler hostname
   ```

   Here, *`hostname`* is the host name of your computer.

   After compilation, the analysis is submitted to a server and returns a job ID. To run a Code Prover analysis, use `polyspace-code-prover` instead of `polyspace-bug-`

`finder`. You can run the `polyspace-code-prover` command with a Polyspace Bug Finder license only, provided you use the `-batch` option.

3. See the status of the current job.

```
polyspace-jobs-manager listjobs -scheduler hostname
```

   You can locate the current job using the job ID.

4. Once the job is completed, you can explicitly download the results.

```
polyspace-jobs-manager download -job jobID -results-folder .
     -scheduler hostname
```

   Here, *`jobID`* is the job ID from the submission.

The results folder contains the downloaded results file (with extension `.psbf`). Open the results in the user interface of the desktop product, Polyspace Bug Finder.

See Also

More About

- “Install Products for Submitting Polyspace Analysis from Desktops to Remote Server” on page 1-14
- “Send Polyspace Analysis from Desktop to Remote Servers”
- “Send Polyspace Analysis from Desktop to Remote Servers Using Scripts”

4

Polyspace Bug Finder and Polyspace Code Prover

Choose Between Polyspace Bug Finder and Polyspace Code Prover

Polyspace Bug Finder and Polyspace Code Prover detect run-time errors through static analysis. Though the products have a similar user interface and the mathematics underlying the analysis can sometimes be the same, the goals of the two products are different.

Bug Finder quickly analyzes your code and detects many types of defects. Code Prover checks *every* operation in your code for a set of possible run-time errors and tries to prove the absence of the error for all execution paths[1]. For instance, for *every* division in your code, a Code Prover analysis tries to prove that the denominator cannot be zero. Bug Finder does not perform such exhaustive verification. For instance, Bug Finder also checks for a division by zero error, but it might not find all operations that can cause the error.

The two products involve differences in setup, analysis and results review, because of this difference in objectives. In the following sections, we highlight the primary differences between a Bug Finder and a Code Prover analysis (also known as verification). Depending on your requirements, you can incorporate one or both kinds of analyses at appropriate points in your software development life cycle.

How Bug Finder and Code Prover Complement Each Other

1. For each operation in your code, Code Prover considers all execution paths leading to the operation that do not have a previous error. If an execution path contains an error prior to the operation, Code Prover does not consider it. See “Code Prover Analysis Following Red and Orange Checks” (Polyspace Code Prover).

Overview

Use both Bug Finder and Code Prover regularly in your development process. The products provide a unique set of capabilities and complement each other. For possible ways to use the products together, see "Workflow Using Both Bug Finder and Code Prover" on page 4-9.

This table provides an overview of how the products complement each other. For details, see the sections below.

Feature	Bug Finder	Code Prover
Number of checkers	251	28 (Critical subset)
Depth of analysis	Fast. For instance: • Faster analysis. • Easier set up and review. • Fewer runs for clean code. • Results in real time.	Exhaustive. For instance: • All operations of a type checked for certain critical errors. • More rigorous data and control flow analysis.
Reporting criteria	Probable defects	Proven findings
Bug finding criteria	Few false positives	Zero false negatives

Faster Analysis with Bug Finder

How much faster the Bug Finder analysis is depends on the size of the application. The Bug Finder analysis time increases linearly with the size of the application. The Code Prover verification time increases at a rate faster than linear.

One possible workflow is to run Code Prover to analyze modules or libraries for robustness against certain errors and run Bug Finder at integration stage. Bug Finder analysis on large code bases can be completed in a much shorter time, and also find integration defects such as **Declaration mismatch** and **Data race**.

More Exhaustive Verification with Code Prover

Code Prover tries to prove the absence of:

- **Division by Zero** error on *every* division or modulus operation
- **Out of Bounds Array Index** error on *every* array access
- **Non-initialized Variable** error on *every* variable read
- **Overflow** error on *every* operation that can overflow

and so on.

For each operation:

- If Code Prover can prove the absence of the error for all execution paths, it highlights the operation in green.
- If Code Prover can prove the presence of a definite error for all execution paths, it highlights the operation in red.
- If Code Prover cannot prove the absence of an error or presence of a definite error, it highlights the operation in orange. This small percentage of orange checks indicate operations that you must review carefully, through visual inspection or testing. The orange checks often indicate hidden vulnerabilities. For instance, the operation might be safe in the current context but fail when reused in another context.

  You can use information provided in the Polyspace user interface to diagnose the checks. See “More Rigorous Data and Control Flow Analysis with Code Prover” on page 4-7. You can also provide contextual information to reduce unproven code even further, for instance, constrain input ranges externally.

Bug Finder does not aim for exhaustive analysis. It tries to detect as many bugs as possible and reduce false positives. For critical software components, running a bug finding tool is not sufficient because despite fixing all defects found in the analysis, you can still have errors during code execution (false negatives). After running Code Prover on your code and addressing the issues found, you can expect the quality of your code to be much higher. See “Zero False Negatives with Code Prover” on page 4-9.

More Specific Defect Types with Bug Finder

Code Prover checks for types of run-time errors where it is possible to mathematically prove the absence of the error. In addition to detecting errors whose absence can be mathematically proven, Bug Finder also detects other defects.

For instance, the statement `if(a=b)` is semantically correct according to the C language standard, but often indicates an unintended assignment. Bug Finder detects such unintended operations. Although Code Prover does not detect such unintended operations, it can detect if an unintended operation causes other run-time errors.

Examples of defects detected by Bug Finder but not by Code Prover include good practice defects (Polyspace Bug Finder), resource management defects (Polyspace Bug Finder), some programming defects (Polyspace Bug Finder), security defects (Polyspace Bug Finder), and defects in C++ object oriented design (Polyspace Bug Finder).

For more information, see:

- “Defects” (Polyspace Bug Finder): List of defects that Bug Finder can detect.
- “Run-Time Checks” (Polyspace Code Prover): List of run-time errors that Code Prover can detect.

Easier Setup Process with Bug Finder

Even if your code builds successfully in your compilation toolchain, it can fail in the compilation phase of a Code Prover verification. The strict compilation in Code Prover is related to its ability to prove the absence of certain run-time errors.

- Code Prover strictly follows the ANSI® C99 Standard, unless you explicitly use analysis options that emulate your compiler.

  To allow deviations from the ANSI C99 Standard, you must use the options. If you create a Polyspace project from your build system, the options are automatically set.
- Code Prover does not allow linking errors that common compilers can permit.

  Though your compiler permits linking errors such as mismatch in function signature between compilation units, to avoid unexpected behavior at run time, you must fix the errors.

For more information, see “Troubleshoot Compilation and Linking Errors” (Polyspace Code Prover).

Bug Finder is less strict about certain compilation errors. Linking errors, such as mismatch in function signature between different compilation units, can stop a Code Prover verification but not a Bug Finder analysis. Therefore, you can run a Bug Finder analysis with less setup effort. In Bug Finder, linking errors are often reported as a defect after the analysis is complete.

Fewer Runs for Clean Code with Bug Finder

To guarantee absence of certain run-time errors, Code Prover follows strict rules once it detects a run-time error in an operation. Once a run-time error occurs, the state of your program is ill-defined and Code Prover cannot prove the absence of errors in subsequent code. Therefore:

- If Code Prover proves a definite error and displays a red check, it does not verify the remaining code in the same block.

  Exceptions include checks such as **Overflow**, where the analysis continues with the result of overflow either truncated or wrapped around.
- If Code Prover suspects the presence of an error and displays an orange check, it eliminates the path containing the error from consideration. For instance, if Code Prover detects a **Division by Zero** error in the operation `1/x`, in the subsequent operation on `x` in that block, `x` cannot be zero.
- If Code Prover detects that a code block is unreachable and displays a gray check, it does not detect errors in that block.

For more information, see “Code Prover Analysis Following Red and Orange Checks” (Polyspace Code Prover).

Therefore, once you fix red and gray checks and rerun verification, you can find more issues. You need to run verification several times and fix issues each time for completely clean code. The situation is similar to dynamic testing. In dynamic testing, once you fix a failure at a certain point in the code, you can uncover a new failure in subsequent code.

Bug Finder does not stop the entire analysis in a block after it finds a defect in that block. Even with Bug Finder, you might have to run analysis several times to obtain completely clean code. However, the number of runs required is fewer than Code Prover.

Results in Real Time with Bug Finder

Bug Finder shows some analysis results while the analysis is still running. You do not have to wait until the end of the analysis to review the results.

Code Prover shows results only after the end of the verification. Once Bug Finder finds a defect, it can display the defect. Code Prover has to prove the absence of errors on all execution paths. Therefore, it cannot display results during analysis.

More Rigorous Data and Control Flow Analysis with Code Prover

For each operation in your code, Code Prover provides:

- Tooltips showing the range of values of each variable in the operation.

  For a pointer, the tooltips show the variable that the pointer points to, along with the variable values.
- Graphical representation of the function call sequence that leads to the operation.

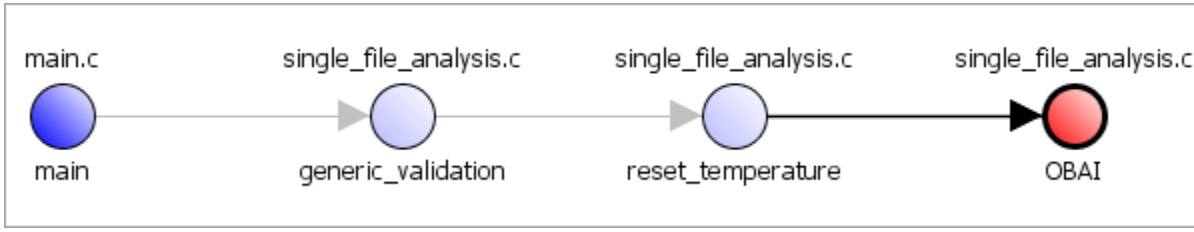
By using this range information and call graph, you can easily navigate the function call hierarchy and understand how a variable acquires values that lead to an error. For instance, for an **Out of Bounds Array Index** error, you can find where the index variable is first assigned values that lead to the error.

When reviewing a result in Bug Finder, you also have supporting information to understand the root cause of a defect. For instance, you have a traceback from where Bug Finder found a defect to its root cause. However, in Code Prover, you have more complete information, because the information helps you understand all execution paths in your code.

```
167 static void Square_Root_conv(double alpha, float* beta_pt)
168 /* Perform arithmetic conversion of alpha to beta */
169 {
170     *beta_pt = (float)((1.5 + cos(alpha)) / 5.0);
171 }
172
173
174 stati
175 {
176     d
177     f
178     f
179
180     Square_Root_conv(alpha, &beta);
181
182     gamma = (float)sqrt(beta - 0.75);   /* always sqrt(negative number) */
183 }
```

Dereference of parameter 'beta_pt' (pointer to float 32, size: 32 bits):
Pointer is not null.
Points to 4 bytes at offset 0 in buffer of 4 bytes, so is within bounds (if memory is allocated).
Pointer may point to variable or field of variable:
'beta', local to function 'Square_Root'.
Assignment to dereference of parameter 'beta_pt' (float 32): [0.1 .. 0.5]
Press 'F2' for focus

Data Flow Analysis in Code Prover



Control Flow Analysis in Code Prover

Few False Positives with Bug Finder

Bug Finder aims for few false positives, that is, results that you are not likely to fix. By default, you are shown only the defects that are likely to be most meaningful for you.

Bug Finder also assigns an attribute called impact to the defect types based on the criticality of the defect and the rate of false positives. You can choose to analyze your

code only for high-impact defects. You can also enable or disable a defect that you do not want to review[2].

Zero False Negatives with Code Prover

Code Prover aims for an exhaustive analysis. The software checks every operation that can trigger specific types of error. If a code operation is green, it means that the operation cannot cause those run-time errors that the software checked for[3]. In this way, the software aims for zero false negatives.

If the software cannot prove the absence of an error, it highlights the suspect operation in red or orange and requires you to review the operation.

Workflow Using Both Bug Finder and Code Prover

If you have both Bug Finder and Code Prover, based on the above differences, you can deploy the two products appropriately in your software development workflow. For instance:

- All developers in your organization can run Bug Finder on newly developed code. For maintaining standards across your organization, you can deploy a common configuration that looks only for specific defect types.

  Code Prover can be deployed as part of your unit testing suite.
- You can run Code Prover only on critical components of your project, while running Bug Finder on the entire project.
- You can run Code Prover on modules of code at the unit testing level, and run Bug Finder when integrating the modules.

  You can run Code Prover before unit testing. Code Prover exhaustively checks your code and tries to prove the presence or absence of errors. Instead of writing unit tests for your entire code, you can then write tests only for unproven code. Using Code Prover before unit testing reduces your testing efforts drastically.

Depending on the nature of your software development workflow and available resources, there are many other ways you can incorporate the two kinds of analysis. You can run both products on your desktop during development or as part of automated testing on a

2. You can also disable certain Code Prover defects related to non-initialization.
3. The Code Prover result holds only if you execute your code under the same conditions that you supplied to Code Prover through the analysis options.

remote server. Note that it is easier to interpret and fix bugs closer to development. You will benefit from using both products if you deploy them early and often in your development process.

There are two important considerations if you are running both Bug Finder and Code Prover on the same code.

- Both products can detect violations of coding rules such as MISRA C® rules and JSF® C++ rules.

  However, if you want to detect MISRA C:2012 coding rule violations alone, use Bug Finder. Bug Finder supports all the MISRA C:2012 coding rules. Code Prover does not support a few rules.
- If a result is found in both a Bug Finder and Code Prover analysis, you can comment on the Bug Finder result and import the comment to Code Prover.

  For instance, most coding rule checkers are common to Bug Finder and Code Prover. You can add comments to coding rule violations in Bug Finder and import the comments to the same violations in Code Prover. To import comments, open your result set and select **Tools > Import Comments**.
- You can use the same project for both Bug Finder and Code Prover analysis. The following set of options are common between Bug Finder and Code Prover:

  -
  -
  -
  -
  -
  -
  - , except

You might have to change more of the default options when you run the Code Prover verification because Code Prover is stricter about compilation and linking errors.