



## Sizing Guide

DOCUMENT VERSION: 3.0 – 2020-05-15

CUSTOMER

RELEASED FOR SAP CUSTOMERS AND SAP PARTNERS

# SAP Plant Connectivity 15.4 (SP00)

THE BEST RUN

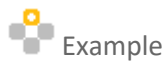


# Disclaimer

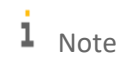
Some components of this product are based on Java™. Any code change in these components may cause unpredictable and severe malfunctions and is therefore expressly prohibited, as is any decompilation of these components.



<Put your caution here>



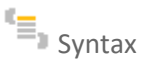
<Put your example here>



<Put your note here>



<Put your recommendation here>



<Put your syntax here>

# Document History

Version	Date	Change
2.0	2019-10-23	Updates for PCo 15.3
3.0	2020-05-15	Updates for PCo 15.4 (Updated to latest template)

# Table of Contents

1	INTRODUCTION .....	6
1.1	Functions of SAP Plant Connectivity.....	6
1.2	Architecture of SAP Plant Connectivity .....	7
1.3	Factors that Influence the Performance .....	8
2	SIZING FUNDAMENTALS AND TERMINOLOGY .....	9
3	HARDWARE REQUIREMENTS AND SIZING .....	10
3.1	Processor (CPU).....	10
3.2	Memory .....	10
3.3	Deployment of PCo .....	11
4	CONFIGURATION OPTIONS INFLUENCING THE PERFORMANCE.....	14
4.1	4.1 PCo-Internal Processes .....	14
4.1.1	<i>Tag-based Notification Processing</i> .....	14
4.1.2	<i>Message Queue and Dispatch Settings</i> .....	14
4.1.3	<i>Multiple Call Destination Systems</i> .....	15
4.1.4	<i>Enhanced Notification Processing</i> .....	16
4.1.5	<i>Enhanced Method Processing</i> .....	16
4.1.6	<i>Tag Query Caching</i> .....	16
4.1.7	<i>Log Settings</i> .....	17
4.2	Data Sources .....	17
4.2.1	<i>Asset Framework Source System</i> .....	17
4.2.2	<i>OSISoft PI Source System</i> .....	17
4.2.3	<i>File Monitor and File System Source System</i> .....	17
4.2.4	<i>Modbus Source System</i> .....	18
4.2.5	<i>MQTT Source System (Subscriber)</i> .....	18
4.2.6	<i>ODBC and OLE DB Source System</i> .....	18
4.2.7	<i>OPC A&amp;E, DA, and HDA Source System</i> .....	18
4.2.8	<i>OPC UA Source System</i> .....	18
4.2.9	<i>Timer Source System</i> .....	19
4.3	Connected Systems (Destination Systems).....	19
4.3.1	<i>Consuming Application System (MII, ME, EWM)</i> .....	19
4.3.2	<i>Data Streaming Destination System</i> .....	19
4.3.3	<i>MII Destination System</i> .....	20
4.3.4	<i>MQTT Destination System (Publisher)</i> .....	20
4.3.5	<i>Multiple Call Destination System</i> .....	20
4.3.6	<i>OData Destination System, RESTful Web Service</i> .....	20
4.3.7	<i>OPC UA Destination System</i> .....	20
4.3.8	<i>Query Destination System</i> .....	20

## Customer

4.3.9	<i>RFC Destination System</i> .....	20
4.3.10	<i>Simulation Destination System</i> .....	20
4.4	<i>PCo as a Server</i> .....	21
4.4.1	<i>SAP MII Query Server</i> .....	21
4.4.2	<i>Web Socket Server</i> .....	21
4.4.3	<i>OPC UA Server</i> .....	21
4.4.4	<i>PCo Web Server</i> .....	21
5	<b>PERFORMANCE OF TYPICAL USE CASES</b> .....	22
5.1	<b>Data Collection Through a Tag-Based Notification</b> .....	22
5.2	<b>Destination System Calls with Response Processing</b> .....	23
5.3	<b>PCo as an OPC UA Server</b> .....	25
5.4	<b>MIQ Queries</b> .....	27
5.5	<b>Web Socket Queries</b> .....	31
6	<b>WINDOWS PERFORMANCE MONITOR AND SAP PLANT CONNECTIVITY PERFORMANCE COUNTERS</b> ...	33
7	<b>COMMENTS AND FEEDBACK</b> .....	34

# 1 INTRODUCTION

This document is intended to identify the factors that influence the performance of your connectivity scenario, to estimate the expected cycle times, and to choose the suitable hardware, configuration, and deployment options for SAP Plant Connectivity (PCo).

## 1.1 Functions of SAP Plant Connectivity

The SAP Plant Connectivity (PCo) component supports various processes. PCo can act as a middleware between various data sources and systems:

- Industry-specific data sources, for example, process control systems, plant Historian systems, and SPC systems
- Business systems, for example, SAP ERP, SAP Manufacturing Execution (ME), SAP MII, SAP EWM, third-party products, databases, for example, SAP HANA, data streaming services
- Other PCo instances

With PCo, you can retrieve and write data tag values from the connected data sources in production either automatically or upon request and forward them to the connected systems. Furthermore, using PCo, you can execute, receive, and process OPC UA method calls and Web service operations, and thereby also map complex coordination tasks in the programmable logic controllers (PLC) environment.

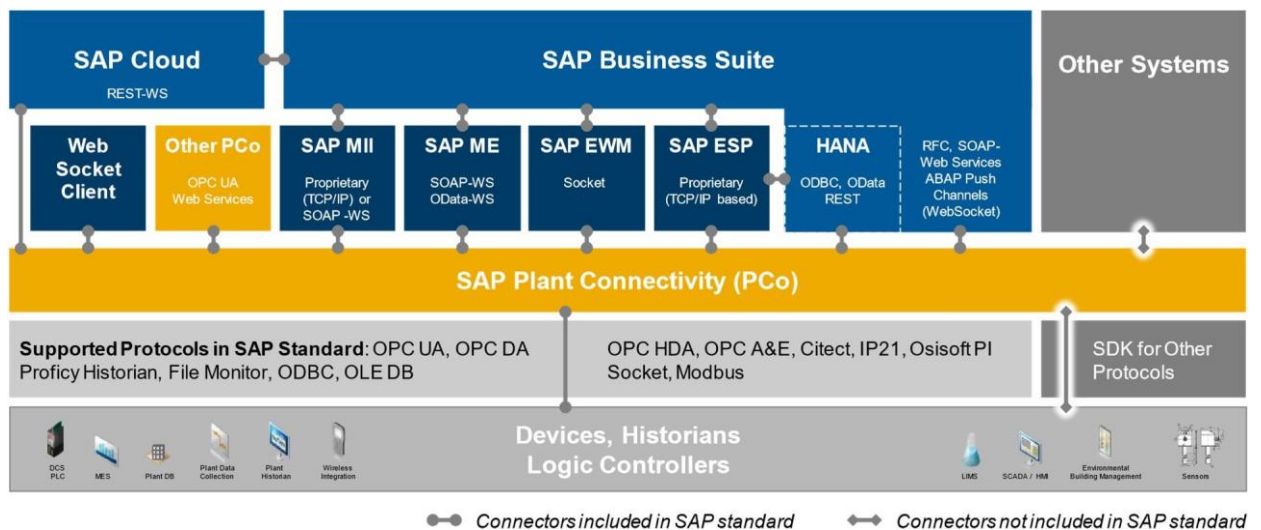


Figure 1: PCo in a Shop Floor Environment

The following diagram shows the maximum extent of connectivity scenarios that can be realized with a single PCo agent instance. PCo actions can be triggered either by clients calling PCo server components (for example, OPC UA methods, Web services, Web sockets, RFC, functions of the query interface) or by events raised from a data source, called tag-based notifications. In both cases, the actions may include one or multiple calls to connected systems and the reading or writing of data tag values from or to a data source.

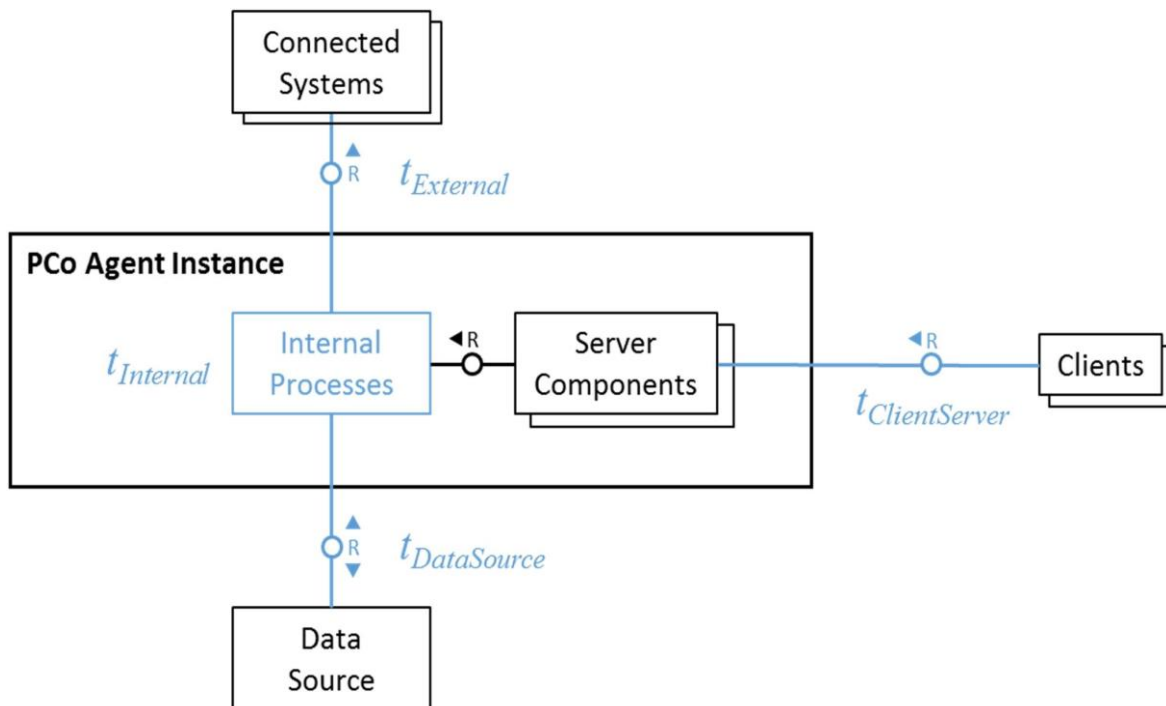


Figure 2: Communication Scenarios

The factors influencing performance and sizing are highlighted in blue. Therefore, the overall performance of a connectivity scenario is determined by the sum of times that are consumed for the following:

- PCo-internal processes
- Communication with connected systems
- Communication with the data source
- Communication from clients to PCo acting as a server A detailed discussion of these factors can be found in chapter 4.

## 1.2 Architecture of SAP Plant Connectivity

SAP Plant Connectivity is a Microsoft .NET application that is installed on Windows PCs. The connectivity tasks are performed by one or multiple **PCo agent instances** that you configure and monitor using the **PCo Management Console**. Every PCo agent instance is a Windows services that can - after being configured - be stopped and started even without the PCo Management Console. Therefore, the PCo Management Console is the central user interface to design, configure, and monitor PCo agent instances, but it is not required to keep the connectivity running once an agent instance has been started.

Therefore, all performance and sizing considerations can be focused on the PCo agent instances. It is the number of PCo agent instances running on your Windows systems in parallel and the connectivity tasks performed by the agent instances that influence the performance of your connectivity scenario.

### **1.3 Factors that Influence the Performance**

When optimizing the performance of a connectivity scenario, the focus areas are hardware and PCo configuration options.

The hardware aspect comprises the selection of an appropriate Windows PC, on which PCo is running, the location of the hardware hosting PCo (installation of multiple PCo agent instances close to the data sources versus a few central PCo installations), and the distribution of connectivity tasks among multiple PCo agent instances.

In addition to the hardware, numerous PCo configuration options influence the performance.

The selection of suitable hardware is discussed in chapter 2 while the most important PCo configuration options are explained in chapter 4.



## 2 SIZING FUNDAMENTALS AND TERMINOLOGY

SAP provides general sizing information on [www.sap.com/sizing](http://www.sap.com/sizing). For the purpose of this guide, we assume that you are familiar with sizing fundamentals. This section explains the most important sizing terms, as these terms are used extensively in this document.

### Sizing

Sizing means determining the hardware requirements of an SAP application, such as network bandwidth, physical memory, CPU processing power, and I/O capacity. The size of the hardware and database is influenced by both business aspects and technological aspects. This means that the number of users using the various application components and the data load they put on the server must be taken into account.

### Benchmarking

Sizing information can be determined using SAP Standard Application Benchmarks ([www.sap.com/benchmark](http://www.sap.com/benchmark)). Released for technology partners, benchmarks provide basic sizing recommendations to customers by placing a substantial load upon a system during the testing of new hardware, system software components, and relational database management systems (RDBMS). All performance data relevant to the system, user, and business applications are monitored during a benchmark run and can be used to compare platforms.

### SAPS

The SAP Application Performance Standard (SAPS) is a hardware-independent unit that describes the performance of a system configuration in the SAP environment. It is derived from the Sales and Distribution (SD) Benchmark, where 100 SAPS is defined as the computing power to handle 2,000 fully business processed order line items per hour. (For more information about SAPS, see [www.sap.com/benchmark](http://www.sap.com/benchmark) → [Measuring in SAPS](#)).

### Greenfield Sizing

Greenfield sizing refers to the sizing approach that provides statements about platform-independent requirements of the hardware resources necessary for representative, standard delivery SAP applications. The initial sizing guidelines assume optimal system parameter settings, standard business scenarios, and so on.

### Expert Sizing

This term refers to a sizing exercise where customer-specific data is being analyzed and used to put more detail on the sizing result. The main objective is to determine the resource consumption of customized content and applications (not SAP standard delivery) by comprehensive measurements. More information can be found [here](#).

### Configuration and System Landscaping

Hardware resource and optimal system configuration greatly depend on the requirements of the customer-specific project. This includes the implementation of distribution, security, and high availability solutions by different approaches using various third-party tools. In the case of high availability through redundant resources, for example, the final resource requirements must be adjusted accordingly.

## Customer

## 3 HARDWARE REQUIREMENTS AND SIZING

If PCo is expected to be the performance bottleneck, you should take the following factors into consideration:

- Processor (CPU) and number of CPU cores: This influences the number of possible threads that PCo can use for notifications or method calls. However, this only helps to handle load peaks. If the input from the data source is constantly coming in faster than the destination can receive notifications, this will result in a constantly growing notification message queue.
- Network interface: Sometimes the network itself becomes the limiting factor. Therefore, you should make sure that the network speed is sufficient. To rule out network issues between the data source and PCo, you should install PCo on the same server as your data source(s).
- Sufficient memory - both RAM and disk space - might be needed for two reasons:
- If a large number of agent instances are running in parallel: Every agent instance usually reserves memory in the range of 30-100 MB RAM at execution time.

If you have high peaks of event numbers per time unit: Unsent messages are buffered in the notification message queue if the destination cannot receive them fast enough. PCo keeps the messages in RAM, but can, in addition, store the unsent message in the local file system for queuing.

### 3.1 Processor (CPU)

A current model desktop PC or laptop or an industry PC is sufficient for many use cases. PCo standalone (without any external consumption of server functions or notifications) can handle up to 120 queries per second or 750 notification messages per second on such a machine. However, if PCo is operating in a more realistic context, for example, with MII or other external systems connected, the overall performance is lower, and the influence of the CPU performance on the overall performance is marginal. A 4-core system is recommended as the baseline. A 4-core system can easily handle 10 messages per second sent to MII in a reliable fashion.

The number of CPU cores determines the maximum number of dispatch threads that can be used for notifications or method calls. This is relevant if you have a high rate of incoming events that need to be sent as messages or that are being processed by the framework. However, you must ensure that the destination system for these messages can also handle that message frequency.

You also need to consider the overall processing capacity of your target system. If you have a cluster of multiple PCo systems sending many messages to one target system, this system might become your bottleneck. In this case, a performance-optimized system setup (for example, maximizing threads) or clustering of multiple target system servers is recommended.

### 3.2 Memory

A full installation of PCo consumes about 100 MB disk space (PCo 15.3.1). Every agent instance reserves about 4 MB of disk space for its event log.

Every agent instance uses approximately 30 to 100 MB of RAM as a base load. There is also an additional amount of about 30 MB of RAM that is shared memory between multiple agent instances. However, this amount of shared memory can be neglected when running a lot of agent instances on the same PCo machine.

If you expect occasional data peaks resulting in a queue build-up, you should reserve more memory - both RAM or - per agent instance. If the number of queued messages is constantly growing, the queue can theoretically use up all available memory. You can limit the memory consumption of the notification message queue (see section 4.1.2).

**Customer**

### 3.3 Deployment of PCo

In theory, it is possible to use one PCo server to collect data from a very large number of data sources that might be distributed over a large physical area (for example, even on the other side of the globe). However, it is a good idea to use multiple PCo servers for several reasons:

- Load balancing
- Network reliability (especially WAN)
- Network performance
- Impact of hardware failure

One option is to define the borders for different PCo systems based on network security areas separated by firewalls, company buildings, production lines, or even single machines (for example, those with high output rates).

Depending on the communication protocol used, the data source that supports PCo agent instances might need to exchange many messages with the source for a single data query. For this reason, SAP recommends installing PCo next to your data sources (for example, on the same network or even on the same machine), to keep network latency to a minimum. Possible deployment options are described in the following figure:

**Customer**

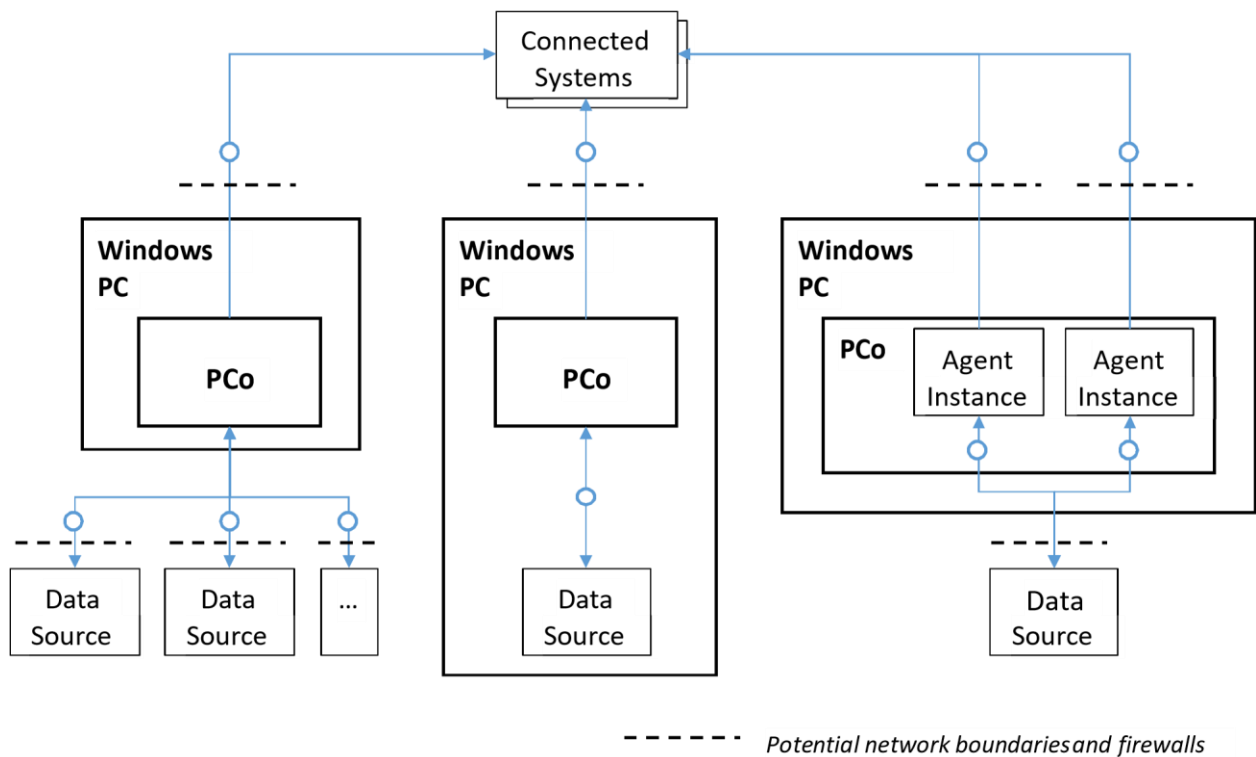


Figure 3: PCo Deployment options

- One PCo for multiple data sources. In this case, PCo usually runs on a different Windows PC than the data source servers. The PCo Windows PC and the data source servers can be in different networks, separated by firewalls (if the source system type allows for it, for example, OPC UA). In this case, network latency is a determining factor for the overall performance. This setup is rather used for small and medium data volumes.
- One PCo per data source. Recommended for high data volume processing, where PCo is installed on the same Windows PC as the data source server.
- Multiple PCo agent instances per data source. It is also possible to connect multiple agent instances to the same data source.

Here are two examples of PCo deployment options from customer implementation projects:

**Customer 1: Automotive Engine Manufacturer Using PCo for Machine Integration**

Three machining lines, one engine assembly line:

- 37 second takt time per engine
- Five Plant Connectivity/Keypware OPC Server platforms
- 6000+ transactions/minute at full rate

**Customer**

**Machining line 1:**

- 89 programmable logic controllers
- 209 PCo agent instances on a single PCo server (4 CPU 16GB RAM 500GB disk)
- 4500 data points Machining line 2:
- 120 programmable logic controllers
- 275 PCo agent instances on a single PCo server (4 CPU 16GB RAM 500GB disk)
- 4700 data points Machining line 3:
- 75 programmable logic controllers
- 167 PCo agent instances on a single PCo server (4 CPU 16GB RAM 500GB disk)
- 2900 data points

**Engine assembly line:**

- 232 programmable logic controllers
- 579 PCo agent instances on two PCo servers (each 8 CPU 16GB RAM 500GB disk)
- 7600 data points

**Customer 2: Vehicle Manufacturer Using PCo for Machine Integration**

One main assembly line, 7 subassembly lines:

- 800 vehicles per day
- 78 second takt time
- One Plant Connectivity/Kepware OPC Server platform
- 1000+ transactions/minute at full rate
- A single PCo instance /4 CPU 8 GB 500GB disk) serving all lines

**Main assembly line:**

- Two programmable logic controllers
- 50 PCo agent instances
- 1000 data points Subassembly lines:
- Five programmable logic controllers
- 20 PCo agent instances
- 400 data poi

**Customer**

## 4 CONFIGURATION OPTIONS INFLUENCING THE PERFORMANCE

### 4.1 4.1 PCo-Internal Processes

Depending on the configuration of the PCo agent instances and the connectivity scenario, a considerable amount of time can be consumed by PCo-internal processes. Depending on the scenario, these processes might be one of the following:

#### 4.1.1 Tag-based Notification Processing

In tag-based notification processing, value changes of data tags at a data source trigger the sending of information to one or multiple destination systems.

The components involved in this process and their effects on the overall performance are as follows:

- Data source capabilities t(DataSource), such as:
  - Refresh rate of the data source. How often can the data source raise a tag change event?
  - Network bandwidth between the data source and the PCo agent instances. Network latencies, especially if the data source and PCo are in different or remote networks.
  - Server capabilities of the data source. Is the data source server able to fire the events in the configured time, or is the server busy?
- PCo-internal processes t(Internal), such as:
  - Message bundling. Instead of calling a destination system with every data change event, you can let PCo collect data in its buffer and send a bundled message to certain destination system types. This reduces the number of calls to a destination system and can have a positive effect on the overall system performance.
  - Message queuing and dispatcher settings (see section 3.1.2)
  - Retry settings. You can configure a notification in such a way that PCo tries to deliver a notification message multiple times in case of a failed delivery. Note that in this case the number of queued notification messages can grow, thus decreasing overall performance.
- Calls to the connected system through the destination system:
  - Network bandwidth between the PCo agent and the connected system
  - Performance of the connected system
  - Type of connected destination system. For example: A Web service has a communication overhead for establishing a connection and authorizes the user with every call, compared to an OPC UA destination that establishes a stable connection.

#### 4.1.2 Message Queue and Dispatch Settings

Queuing and dispatching of notification messages is part of a tag-based notification process. The following settings can have an impact on the performance of a notification process:

[Agent Instance > Notification Processing > Notification Message Queue and Dispatch Settings](#)

**Storage Method:** In a tag-based notification scenario, all notification messages are queued in the memory (RAM) of the PCo host. With the [Storage Method](#) settings, you can define that messages are stored persistently in the local file system in addition to keeping all queued messages in RAM. There are two methods for persistent storage: [Microsoft Message](#)

*Queuing (MSMQ) and FileSystem.* In Windows systems that offer the Windows feature MSMQ, the storage method *Microsoft Message Queuing (MSMQ)* is the preferred option, since it can queue data about six times faster than the storage method *File System*. However, for performance reasons, you should use the storage method *No Storage*, if the loss of queued notification messages after an agent instance restart does not matter.

*Resend Failed Messages Automatically:* With this option, PCo can automatically reprocess message that have been put into the "Message Failures" queue. PCo calculates the number of dispatch threads used for resending failed messages dynamically, so that the timely dispatching of new notification messages coming from the data source is still ensured. However, activating this option can temporarily slow down the dispatching of new notification messages to the destination system.

*Process Notification Messages Exactly Once in Order:* Notification messages are delivered sequentially and in the order in which they were placed in the queue. There is no parallel dispatching of notification messages in this case. This can cause the notification message queue to grow constantly if the receiving system cannot process the notification messages fast enough. Moreover, failed messages may block the sending of any further messages to prevent one of these other messages being delivered first.

*Keep Copies of Queued Notification Messages in Journal Queue:* This setting causes the MSMQ to keep a copy of each notification message in the Journal Messages folder of the MSMQ component. You should only use this setting in the test phase for testing and for tracking errors; otherwise the content of the directory becomes too large and has an adverse effect on performance.

*Make Queued Notification Messages Recoverable:* By default, MSMQ stores notification messages in memory for increased performance, and a notification message may be sent and received from a queue without ever having been written to disk. By selecting the option *Make Queued Notification Messages Recoverable*, you force MSMQ to store a notification message to disk. This increases the system reliability in the case of an unexpected system termination, but can have a negative impact on the overall system performance in scenarios with very high tag change rates.

*Max. Dispatch Threads:* This specifies the maximum number of threads used per CPU core for the parallel sending of notification messages to the destination system. At the runtime of the agent instance, the system only provides a limited number of possible parallel threads. The parallel threads are also used by other PCo processes, such as session handling for OPC UA functions. Therefore, you should not use the maximum value of 250 for the dispatch threads. Moreover, the possible message throughput of the destination systems used determines the number of dispatch threads that can be used with meaningful effect.

For example, a connection to a Web server using a universal Web service destination system only provides a limited number of open HTTP connections. A large number of dispatch threads used that have a relatively small number of usable HTTP connections, can lead to a system load that is too high and can lead to long waiting times when agent instances are stopped.

The Max. Dispatch Threads field is not relevant to the delivery type Exactly Once in Order and so you cannot change it. In the case of the delivery type Exactly Once in Order, delivery is always synchronous and sequential. It is recommended to keep the proposed value of 10 threads per CPU core.

#### **4.1.3 Multiple Call Destination Systems**

The performance of a multiple call destination system is mainly determined by the destination systems that are called from within a multiple call destination system. Calling a multiple call destination system that does not call any destination systems, but, for example, - only performs some variable calculations and branching condition evaluations, does not consume any significant resources.

Example: Multiple call destination system MCD1 calls multiple call destination system MCD2 in a loop 1,000 times. MCD2 calculates the logarithm of a value. The measurement results showed that calling MCD1 plus the loop over MCD2 took only about 1 ms in total.

*General Settings > General Settings > Exclusive Lock*

This option ensures that only a single dispatch thread time can process the multiple call destination within the same agent instance at one point in time. This can ensure, on the one hand, data integrity if the destination systems called within the multiple call destination must be called in a given order. For example, a StartSfc has to be called before a CompleteSfc for the same SFC number. On the other hand, if this sequence does not matter, settings the *Exclusive Lock* prevents can prevent the parallel processing of dispatch threads.

#### **4.1.4 Enhanced Notification Processing**

*Agent Instance > Notification Processing > Enhanced Notification Processing*

Enhanced Notification Processing (ENP) enables customers to control and document the production flow by calling destination systems and accessing source system data in a very flexible way. A typical use case is to connect an SAP ME system to PCo, call an SAP ME Web service from PCo, evaluate the Web service result, and call another Web service depending on the results of the previous Web service calls.

PCo provides an interface that customers can implement. The customer-owned enhancement is represented by a destination system, so that the customer-owned logic is called as part of a notification process. SAP delivers an ENP implementation called *Destination System Calls with Response Processing*.

While an ENP implementation can increase the flexibility of production processes, it can have a negative impact on the system performance if it involves numerous Web service calls and other time-consuming operations.

The standard ENP implementation *Destination System Calls with Response Processing* processes the destination system calls in a sequential and synchronous manner. This means that all the request and response times for every destination system call add up. See section 5.2 for a detailed example.

#### **4.1.5 Enhanced Method Processing**

When you configure a PCo agent as an OPC UA or a Web server, you can use assemblies with predefined methods.

SAP provides two standard Enhanced Method Processing (EMP) implementations *Locking* and *KvpBuffer*. Both assemblies have a small footprint on the overall agent performance. , Performance is only influenced by the data access speed to the local file systems if you use the *KvpBuffer* assembly for writing buffer entries in a persistent manner (meaning that data records are stored on your local file system).

However, customer-owned or third-party EMP implementations can contain complex algorithms that might have a negative effect on performance.

#### **4.1.6 Tag Query Caching**

*Agent Instance > Tag Query > Cache Settings > Cache Mode:*

If set to *Access to Data Source Only*, PCo always first retrieves the full path to the requested tag before reading the tag. Using the other modes (*Access to Cache, to Data Sources as Required, Access Using Aliases Only, Access to Cache Only*) can increase performance. Since the cache mode would load the whole tag structure into memory, SAP recommends that you use the *Access to Cache, to Data Sources as Required* mode for large tag structures (load tag structure on demand into cache).

Section 4.4 MII Queries shows the performance differences between the tag query caching modes.

You can connect from SAP MII to PCo either through a PCo Connector or an UDC Connector. If you define SAP MII tag queries with filters, you must pay attention to the number of data tags at the data source. A large number of data tags can cause the filtering process to take a lot of time. This is especially true for the UDC Connector. Therefore, SAP recommends that you use the PCo Connector. The PCo connector also provides native filters that are specially optimized for specific PCo data sources, for example, for OSISoft PI, OPC UA, and OPC DA.



#### 4.1.7 Log Settings

[Agent Instance](#) > [Host](#) > [General Settings](#) > [Log Level](#)

For high performance, do not use the [Verbose](#) or [Information](#) log levels. If you use these log levels, PCo writes a lot of data to the log that slows down performance significantly.

## 4.2 Data Sources

The performance of communication with tag-based data sources, for example, a PLC or a file system, from which PCo can read data or to which it can write data, is mainly influenced by the following factors:

- Maximum refresh rate and event frequency that the data source can deliver, update rates
- Performance of the server on which the data source is installed
- Network load and bandwidth, if data source is on a remote server
- Number of subscribed data tags
- Single or mass data read/write: Especially with MII transactions
- Security settings (transport and message encryption): Session-based communication vs. stateless communication. With stateless communication, certificate-based communication is very expensive, since all communication involves cryptographic operations with large key sizes, whereas in session-based communication, this overhead occurs only on startup and periodically after session invalidation. A small overhead due to message security cannot be prevented.

Depending on the communication protocol used, the data source that supports PCo agent instances might need to exchange many messages with the source for a single data query. For this reason, SAP recommends that you install PCo next to your data sources (for example, on the same network or even on the same machine), so that minimal network latency is expected. Possible deployment options are described in section 3.3.

The following sections describe specific settings of the data sources that need special attention.

#### 4.2.1 Asset Framework Source System

[Settings](#) > [Events](#) > [Max. Number of Events](#): This denotes the maximum number of events during the update rate interval that will be processed. If this number is too big, the performance can go down, because the source system has to process too many events. On the other hand, if this number is too small, some events might be lost.

#### 4.2.2 OSISoft PI Source System

[Multithreading](#): Number of worker threads (6 is default and recommended). Can parallelize incoming queries. If you use a value of 1, queries are processed sequentially.

#### 4.2.3 File Monitor and File System Source System

Both the file monitor and the file system source systems have a configuration option named [Monitoring Frequency](#). This option specifies how often the folder to monitor is checked. New or changed files in the folder to monitor can only be detected in that frequency. A lower value (meaning a higher check frequency) can lead to a higher system load through a high utilization of disk drives, even if there are no new or changed files. The default value of 1000 ms is a good trade-off between system load and responsive event processing.

If you access a remote file system, network latency and additional time for authentication may occur.

#### 4.2.4 Modbus Source System

The access to a Modbus data source can be made either through a TCP-based or a serial connection. The channel type *Serial* is slower and less reliable than the type *TCP*.

The configuration option *Minimum Update Rate* specifies how often data is to be read by the connected device or the minimum time that has to pass before the Modbus source system sends the next value change to PCo. A low update rate consumes a lot of network bandwidth. The default value of 20 ms is a good trade-off between data accuracy and performance. Choose the update rate of the data tags according to your real needs.

*Tag Definition > etails > Address*: Modbus can read up to 123 registers (about 256 bytes) with a single command. If you have a lot of data tags that span over more registers, PCo needs to issue more than one command to read or write the data, resulting in higher network load. Therefore, it would be preferable to have, for a single notification, all tags in a single address area. Also, with regards to data consistency, you should have tags that correlate with each other in a single address area.

*Minimum Update Rate (of data tags)*: A low update rate consumes a lot of network bandwidth. The default value of 20 ms is a good trade-off between data accuracy and performance. Choose the update rate of the data tags according to your real needs.

#### 4.2.5 MQTT Source System (Subscriber)

See section 4.3.4.

#### 4.2.6 ODBC and OLE DB Source System

Every call to the source system is done in single transactions. Mass operations are not supported

#### 4.2.7 OPC A&E, DA, and HDA Source System

*OPC DA Settings > Settings > Synchronous Read Source* (OPC DA only): When setting up a query scenario in PCo, you should use the option *From Cache*, which results in better performance.

*Settings > Synchronous Read Source (only DA): From Device* is less performant than *From Cache*, but sometimes when the reaction time is critical, it is needed to read tags from the device.

*Group Settings > Update Rate (only DA)*: Default is one second. If you want faster updates, this parameter must be set accordingly.

*HAD Settings > Sampling Interval (In Seconds)*: Determines how often the HDA data source can trigger notifications. The highest rate is once per second.

*HAD Settings > Batch Processing Flag*: When more than one item is to be read/written, the items can be processed by the server in batch, with a single call. In this case the performance is better, but some servers do not support this feature. The recommendation is to set the flag and check whether the multiple read/write query is working properly.

#### 4.2.8 OPC UA Source System

*Transport settings (XML/TCP)*: Although XML-based communication provides potentially a higher compatibility with nonOPC-based applications, it requires much more data on the wire, leading to a performance penalty. *OPC UA > Session*:

*PackageSize for Read/Write/Browse Service*: These settings should be set to 0, which means no size restriction for packages. In other words, PCo always attempts to perform the request in one operation. If you need to reduce the package size for any such operation for compatibility reasons, try to use as large values as possible.

*Subscription > Subscription Parameters*: preferable settings with regard to performance may depend to a large extent on server capabilities. Usually a good strategy is to try to enlarge the message sizes while simultaneously reducing the number of messages, for example, by putting several subscription items into a single notification. In case of subscriptions, some care has to be taken with this approach, though. With

PCo, it is not possible to set the scan rate of individual monitored items. Instead, the scan rate for all items is automatically set to the event interval, which can only be set on source system granularity. If you intend to subscribe to items with different scan rates you should consider creating several source systems with corresponding agent instances.

SAP-internal measurements showed that PCo can easily handle 500 changes to 10 OPC UA data tags per second. (PCo and OPC UA server on the same Windows PC, delays due to sending to destination systems not taken into consideration).

#### **4.2.9 Timer Source System**

The Timer Source System is edge component, that provides the possibilities to call Notification Process for user-defined Timers on schedule. Values for subscription items contain the current event time.

Timer Source System has a configuration options on the two tabs. All options affect its behavior. The main option *Time Interval*. This option specifies how often the timer event will be called. The default value is 60000 ms (1 minute).

Options *Start Time* and *End Time* are set in the local time zone, but stored in the configuration file in UTC time zone. If you import configuration file from another time zone or restart application after changing local time zone you will see these options in the current local time zone. UTC time in the configuration file remains unchanged.

### **4.3 Connected Systems (Destination Systems)**

PCo destination systems are adapters to connected systems to which PCo can send data and - in some cases - from which it can receive responses. Examples of connected systems are: Web servers, OPC UA servers, SAP business systems, databases, other PCo agent instances.

General factors that can influence the communication performance are:

- Characteristics of the protocol being used. For example: Web services (stateless -> establish connection, authorize, encrypt payload, transmit payload, close connection) vs. OPC UA (stateful connection)
- Synchronous vs. asynchronous processing: Asynchronous processing is, per se, not faster than synchronous processing, but PCo is not blocked while processing the request and response.
- Network load and bandwidth, remote
- Security settings
- General performance of the connected systems

#### **4.3.1 Consuming Application System (MII, ME, EWM)**

With respect to the consuming application systems, you should ensure that these systems are also optimized for performance. In particular, watch out for settings that influence the parallel execution of tasks (for example, maximum threads). For more information, see the relevant performance guide.

#### **4.3.2 Data Streaming Destination System**

If you configure a data streaming destination system as destination of a notification, each notification message is mapped to the data stream or data window of the project in the data streaming system and results in a new row. The performance of the transfer to the data streaming destination system mainly depends on the network latency and on the width of the data stream or data window. It can be increased by using the option for message bundling, since this reduces the communication overhead. Make sure that each transferred row can be uniquely identified by its key, either by using a timestamp or an automatically generated identifier (GUID) as key field.

Typically, the transfer time to a data streaming destination system is about 5 milliseconds per row for a row with 10 elements.

### 4.3.3 MII Destination System

*Asynchronous Mode (MII Destination Setting)*: If this mode is active, MII will process the messages asynchronously.

If possible, you should use the PCo message bundling function together with an MII destination in order to reduce the total number of calls to MII.

### 4.3.4 MQTT Destination System (Publisher)

*Client > Keep Alive Interval*: longer interval is better, 0 means that no keep-alive messages will be sent.

*Payload > QoS (Quality of Service)*: 0 – *At Most Once* is the best for performance, but 2 - *Exactly Once* is the best for reliability. A tradeoff must be found.

### 4.3.5 Multiple Call Destination System

See section 4.1.3 *Multiple Call Destination Systems*.

### 4.3.6 OData Destination System, RESTful Web Service

#### 4.3.6.1 Destination System, Web Service Destination System, Universal Web Service Destination System

Connections to Web servers are stateless. This means that every call of a Web service includes establishing a connection, user authorization, message payload transfer, and disconnecting. As a rule of thumb, a minimum time penalty of 100 ms per Web service call can be assumed under optimum network conditions. Calling remote Web services takes roughly 300 to 500 ms per call (see section 5.2 for an example).

Therefore, in most cases, Web service calls are the limiting factors when calculating the maximum event rate that a PCo connectivity scenario can handle. If you call multiple Web services in a sequence within a multiple call destination, the times add up.

### 4.3.7 OPC UA Destination System

Session settings do not influence performance when calling OPC UA methods. However, security settings can have an impact on the performance just like for an OPC UA source system (see section 4.2.8).

### 4.3.8 Query Destination System

A query destination system is always related to a running agent instance that is, in turn, connected to a data source. Therefore, the read and write performance of a query destination is mainly determined by the capabilities of the data source (see section 4.2).

The configuration option *Asynchronous Store Query* means the caller does not wait until the write operation is finished, but it does not speed up the write operation itself.

### 4.3.9 RFC Destination System

Since SAP does not deliver standard implementations of connectivity scenarios between SAP PCo and SAP Business Suite systems, the performance of RFC connections is mainly determined by processing logic implemented in the SAP Business Suite.

### 4.3.10 Simulation Destination System

Simulation destination systems are mainly used for testing purposes. In a productive environment, a simulation destination system could be used, for example:

- To write XML files to a file system, so that other systems can process these files. In this case, the performance is mainly determined by the capabilities of the file system.

- To realize a waiting time within a multiple call destination. Here, of course, the system processing is halted until the waiting time has elapsed.

#### 4.4 PCo as a Server

PCo is acting as a server. For example: Web server, OPC UA server, query server for MII, Web socket, RFC server. Factors influencing the performance are:

- Characteristics of the protocol being used. For example: Web services (stateless → establish connection, authorize, transmit payload, close connection) vs. OPC UA (stateful connection)
- Synchronous vs. asynchronous processing of method calls
- Security settings

##### 4.4.1 SAP MII Query Server

If you are using an SAP MII system of release 12.2 or higher for the query process in connection with PCo, you should choose the *PCo Connector* function in SAP MII rather than the outdated *UDC Data Server*. On PCo side, you select the *SAP MII Query Server* checkbox for an MII PCo connector, but not the *SAP MII Query Server (Before 12.2)* checkbox which sets up PCo as an UDC data server. The PCo connector provides a better performance than the UDC data server. Some detailed numbers about the expected performance can be found in section 5.4.

##### 4.4.2 Web Socket Server

The Web Socket Server in PCo supports both the query scenario and the notification scenario. Some detailed numbers about the expected performance for queries can be found in section 5.5.

The Web Socket Server also allows you to push data to a client that subscribes for tag changes. This notification scenario is started by the client by subscribing to some tags. In case of data changes, PCo pushes the data changes of the tags to the client with a specific notification message. Depending on the update rate of the data source, PCo might push lots of messages to the subscribing client.

If the client is a browser-based UI, it is important that the update of the data source is selected appropriately to prevent an overload of the UI. It is recommended that you set update rates in the data source that are greater than or equal to one second.

##### 4.4.3 OPC UA Server

If your OPC UA client allows multiple OPC UA methods to be called in a single request, this is the preferred way.

In addition, the performance impact of the security settings applies in the same way as for the OPC UA source system (see section 4.2.8).

##### 4.4.4 PCo Web Server

When PCo is acting as a Web server, the same considerations apply as for Web service destination systems (see section 4.3.6). Generally speaking, if you have the choice between PCo acting as an OPC UA server or a Web server, you should opt for the OPC UA option due to less communication overhead.

## 5 PERFORMANCE OF TYPICAL USE CASES

### 5.1 Data Collection Through a Tag-Based Notification

In this scenario, a PCo agent instance reacts to changes of data tag values at an OPC UA server. The data tag value is then sent to SAP Manufacturing and Intelligence (MII). Within MII, a PCo query is initiated that sets another data tag value at the same OPC UA server as an acknowledgement.

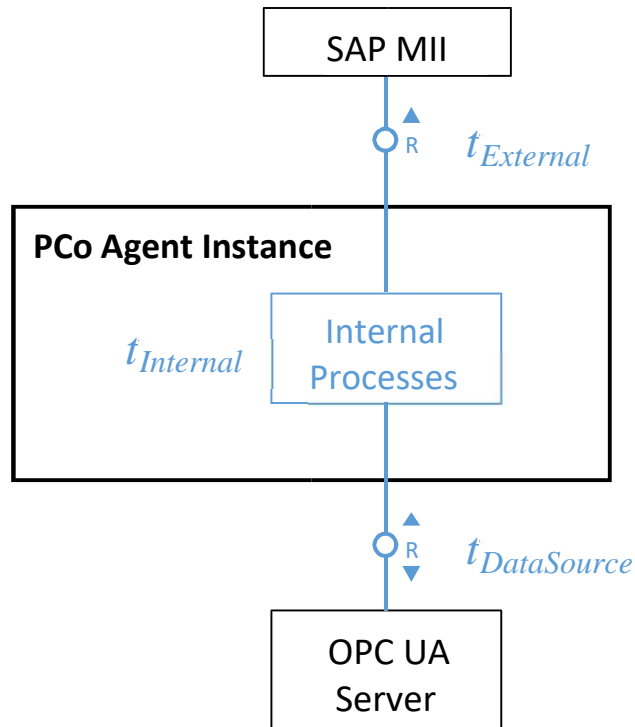


Figure 4: Scenario Block Diagram

This scenario involves the following PCo and MII configuration entities:

- PCo
  - OPC UA source system, acting as an OPC UA client
  - An agent instance with a subscription item that reacts to the changing of a single OPC UA data tag
  - PCo notification message queue (set to storage method *No Storage*)
  - Destination system of type *MII Destination*, which connects to an MII transaction
- MII
  - PCo connector, connecting to the PCo agent instance

- A PCo query that writes a data tag value to the OPC UA server to which the PCo agent instance is connected
- A transaction that encapsulates the PCo query

A complete roundtrip is measured from the changing of the first data tag value (initiated manually) until the changing of the second data tag value (performed by MII).

The measurement yielded roundtrip times of 2 to 3 seconds. PCo and MII ran on different servers, but within the same company network.

The results show clearly that the connectivity in this scenario has a quite limited throughput. The limiting factor is the communication between PCo and MII.

If you have a use case where the write-back to PCo is not required, but only high amounts of data coming from a data source at a high frequency are to be sent to SAP MII, it is recommended that you use the PCo message bundling functionality. PCo would then collect the single tag change value in the message queue, and then send a single message that contains all the single tag change values to MII. This reduces the communication effort between PCo and MII.

## 5.2 Destination System Calls with Response Processing

A typical use case for PCo is to react to data tag value changes at a data source that triggers the call of a Web service. The result of the Web service is then written back to the data source. The whole process is tag-based. That means that the value change of a *trigger tag* initiates the process. The data sent to the Web server also comes mainly from data tags. At the end of the roundtrip, the trigger tag is reset to its original value, so that the next roundtrip could be initiated. In addition, a *response tag* changes its value from 0 to 1, signaling the end of the process.

The data source could be the logic controller of a machine requesting information from SAP ME or SAP MII. The machine would toggle the *trigger tag*, thereby starting the process, and then wait until PCo toggles the *response tag*. The machine would then read any response of the Web service from the data tags of the logic controller. The following set-up was chosen for testing this scenario:

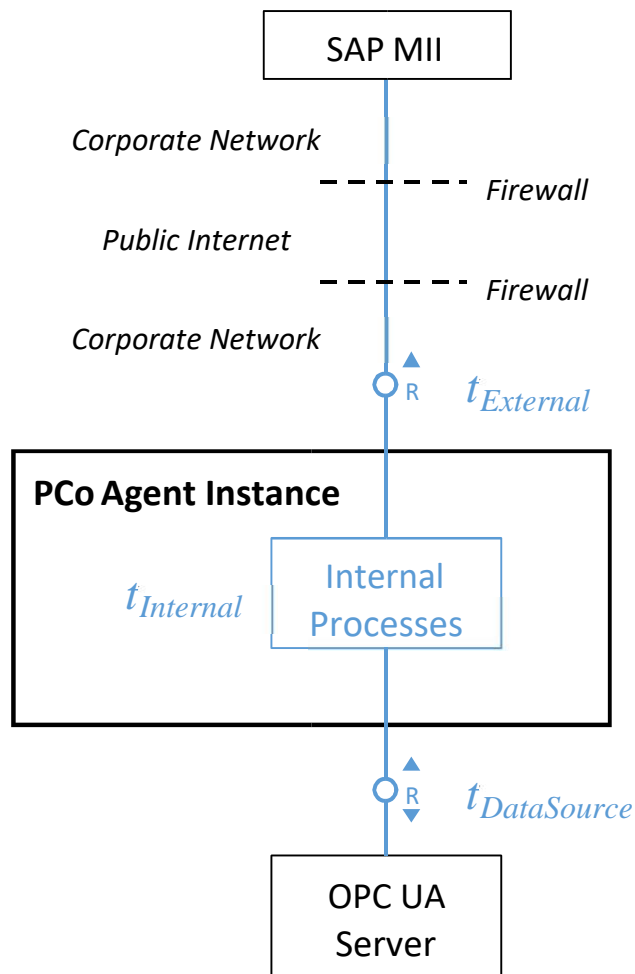


Figure 5: Scenario Block Diagram

This scenario involves the following components:

- Tag-based OPC UA server, providing a trigger tag, a response tag, two data value tags (one whose value is sent to the Web service, the other that receives the Web service call result)
- PCo
  - OPC UA source system, acting as an OPC UA client that connects to the OPC UA Server
  - An agent instance with two subscription items that react to the changing of trigger tag and another that transfers the data value tag to the PCo notification
  - PCo notification message queue (set to storage method *No Storage*)
  - Agent uses *Enhanced Notification Processing* with the standard implementation *Destination System Call with Response Processing*. The standard implementation takes care of managing the Web service call and setting the data tag values at the OPC UA server.
  - Destination system of type *Universal Web Service Destination*, which calls a WSDL-based Web service in SAP MII.



- SAP MII
  - A transaction that receives a numeric value and returns a numeric value. The transaction is called by a Web service.

PCo and SAP MII are located in different networks, even in different countries (Germany and Switzerland). The communication between PCo and MII is routed through the public Internet.

A roundtrip starts with the trigger tag changing at the data source and ends with PCo changing the response tag.

The measurements for a roundtrip yielded results between 350 and 500 ms. The biggest portion of the roundtrip time came from the Web service call which varied between 320 and 450 ms, whereas the time consumption of PCo-internal processes only varies between 20 and 50 ms.

The duration of the Web service call is mainly caused by the connection setup and teardown for every Web call and the varying network latency of the public Internet. MII-internal processing times are included in the total Web service processing time, although they were quite low in this example, since the transaction used was quite simple.

### 5.3 PCo as an OPC UA Server

PCo can act as an OPC UA server, offering OPC UA methods that can be called by an OPC UA client. PCo also provides a client, which is, for calling methods, the OPC UA destination system. Since PCo provides both the server and the client, two PCo agent instances can communicate with each other using OPC UA methods.

In the following scenario, the processing time for calling PCo OPC UA methods is investigated, An OPC UA client calls a method that is provided by a PCo agent instance that is acting as an OPC UA service. The call is routed to a multiple call destination (PCo-internal process). Within the multiple call destination, a very simple calculation happens. The result of the calculation is passed back to the OPC UA client.

For testing purposes, the client is an OPC UA destination system that is called by a multiple call destination system 100,000 times. The multiple call destination system is called through its own testing function.

Therefore, the total performance is mainly determined by the following factors:

1. Client:
  - Multiple call destination, calling an OPC UA destination system 100,000 times in a loop. Includes evaluation of a branching condition, calling the OPC UA destination system, and variable calculations.
  - OPC UA destination system, calling an OPC UA method with one input and one output parameter on the server side.
2. Server:
  - OPC UA server, providing a single method (direct destination call, no enhanced method processing), receiving the call from the client and forwarding it to the PCo internal processes.
  - A notification that converts the method input and output parameters into the input and output variables of a multiple call destination, and finally calls the multiple call destination.
  - A multiple call destination that receives a numeric value as an input, increments the numeric value by 1, and passes it back to the notification as an output variable.

The goal was to measure the highest OPC UA method call throughput possible by eliminating every factor that could slow down the process, for example:

- Network effects (client and server were running on the same PC)
- Detailed log level

- OPC UA communication security settings

The OPC UA client and the OPC UA server (which is the PCo agent instance) are located on the same Windows PC.

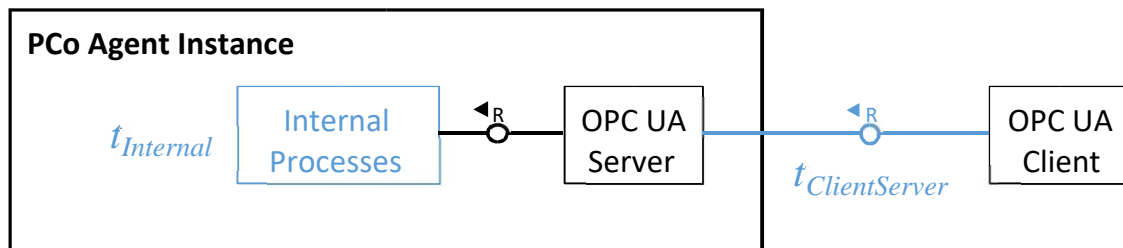


Figure 6: Scenario Block Diagram

The test set-up is as follows:

PCo:

- Hardware: Intel(R) Core(TM) i5-4300U CPU @ 1.90GHz, 2501 Mhz, 2 Core(s), 4 Logical Processor(s)
- Operating system: Microsoft Windows 10 Enterprise
- Total physical memory: 16 GB
- PCo 15.1 SP04
- Log level set to "Error"
- Agent instance running as Windows service
- OPC UA server endpoint no security or encryption Network:
- No network involved; client and server running on the same machine

The following tables show the results of the measurements:

Running agent instance without any load put on it ("idle mode"):

Average CPU load	1%
Average memory consumption	53.0 MB

The numbers in the table above show the CPU load and memory consumption that is required to keep an OPC UA server running in idle mode.

The following tables show the measurement results for putting the highest load possible on the OPC UA server. Therefore, the CPU load is always above 100%.

Total time for 100000 OPC UA method calls, including operations on client and server side	91258.5 ms
Time for a single OPC UA method call, including operations on client and server side	0.91 ms
Average CPU load	> 100%
Average memory consumption	57.0 MB

The total throughput time for a single call, starting from the multiple call destination at the client to the multiple call destination at the server and back is below 1 ms. This includes client-server communications and variable and branching condition processing within the two multiple call destination systems. Therefore, one can say that communication through OPC UA client and server components is very fast and efficient.

#### 5.4 MII Queries

PCo is typically used for reading and writing data from SAP MII to a tag-based data source. The user would configure a PCo agent instance acting as an SAP MII query server. In MII, a PCo connector is set up which provides the connection to PCo.

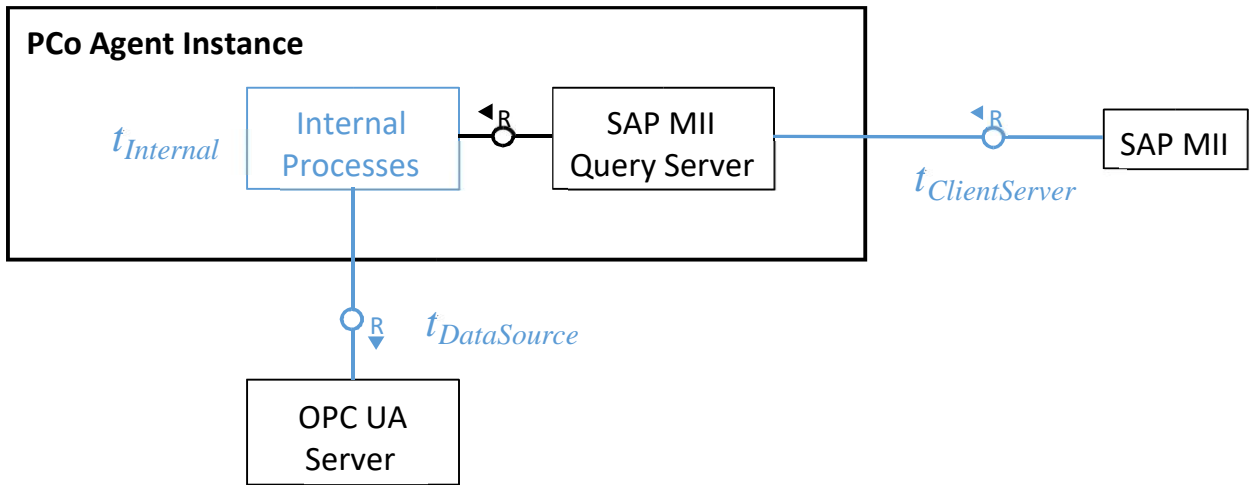


Figure 7: Scenario Block Diagram

In this scenario, PCo and the OPC UA server are installed on the same PC. MII is located in the same network as the PCo PC. Therefore, effects coming from long-distance network connections are not reflected in the test results. The test set-up is as follows: PCo:

- Hardware: Intel(R) Core(TM) i5-4300U CPU @ 1.90GHz, 2501 Mhz, 2 Core(s), 4 Logical Processor(s)
- Operating system: Microsoft Windows 10 Enterprise
- Total physical memory: 16 GB
- PCo 15.1 SP03
- Log level set to "Error"
- Cache Mode Access to Cache, to Data Source as Required
- Agent instance running as Windows service
- Agent instance without subscription items Data source:
- Tag-based OPC UA server
- No security policies for the TCP/IP connection
- OPC UA server and PCo installed on the same Windows machine SAP MII:
- MII 15.1 SP3 Patch 0
- PCoConnector
- TagFixedQuery Network:
- SAP company network

The tests were performed using MII transactions that read and write 1, 10, and 100 tags from and to PCo 10,000 times. Every measurement was repeated 10 times to calculate an average. The following tables show the results of the measurements:

Running agent instance without any load put on it ("idle mode"):

Average CPU load	< 0.1%
Average memory consumption	31.0 MB

CPU load and memory consumption in idle mode depend on the source system type. In this example, the OPC UA source system monitors and keeps the session between the OPC UA server and PCo alive, which in total does not consume a lot of system performance.

The following tables show the measurement results for reading and writing 1, 10, and 100 tags at the same time at the highest frequency possible. Although the total time for reading and writing more than one tag is higher than for a single tag only, the results show that it is more efficient - from a per-tag perspective - to read and write multiple data tags at the same time.

The examples are intended to give you an idea of the read and write frequency that PCo and SAP MII could handle. The reason for the significant CPU loads is the high read and write frequency in the examples above. If you have less than 5 read and write operations per second, there should be no significant CPU load with the default PCo settings.

**Read:**

	1 Tag	10 Tags	100 Tags
Total time for 1000 read operations	20900 ms	22790 ms	58797 ms
Time per read operation	20.9 ms	22.8 ms	58.8 ms
Time per read operation per tag	20.9 ms	2.28 ms	0.588 ms
Average CPU load	11.3 %	13.1 %	14.1 %
Peak CPU load	23.2 %	27.5 %	26.8 %
Average memory consumption	32.6 MB	32.8 MB	33.2 MB

**Write:**

	1 Tag	10 Tags	100 Tags
Total time for 1000 write operations	20926 ms	22800 ms	41996 ms
Time per write operation	20.9 ms	22.8 ms	42.0 ms
Time per write operation per tag	20.9 ms	2.28 ms	0.420 ms

Average CPU load	11.5 %	9.2 %	13.2 %
Peak CPU load	25.0 %	20.2 %	26.6 %
Average memory consumption	32.7 MB	33.2 MB	34.9 MB

Variations:

Read 100 Tags

	Log level <i>Error</i>	Log level <i>Verbose</i>
Total time for 1000 read operations	58797 ms	60816 ms
Time per read operation	58.8 ms	60.8 ms
Average CPU load	14.1 %	24.0 %
Peak CPU load	26.8 %	70.1 %
Average memory consumption	33.2 MB	34.5 MB

Setting the agent log level from the default setting *Error* to *Verbose* impacts the CPU load, but has no significant performance loss for a single agent. On the other hand, if more agent instances run on log level *Verbose*, this could slow down the whole PC.

Read 100 Tags

	Query cache mode <i>Access to Cache, to Data Source as Required</i>	Query cache mode <i>Data Source Only</i>
Total time for 1000 read operations	58797 ms	460000 ms
Time per read operation	58.8 ms	460 ms

Customer

Average CPU load	14.1 %	90.2 %
Peak CPU load	26.8 %	113.9 %
Average memory consumption	33.2 MB	32.4 MB

Default cache mode *Access to Cache, to Data Source as Required* is significantly faster and puts less load on the CPU than the mode *Data Source Only*. Therefore, it is recommended that you use the default query cache mode whenever it is possible.

## 5.5 Web Socket Queries

PCo is typically used for reading and writing data from a browser-based UI. The user would configure a PCo agent instance acting as an SAP Web socket server.

In this scenario, PCo and the OPC UA server are installed on the same PC. The Web socket test UI is located in the same network as the PCo PC. Therefore, effects coming from long-distance network connections are not reflected in the test results.

The test set-up is as

follows: PCo:

- Hardware: Intel(R) Core(TM) i7-6820HQ CPU @ 2.70GHz, 2.71 GHz, 4 Core(s), 8 Logical Processor(s)
- Operating system: Microsoft Windows 10 Enterprise
- Total physical memory: 32 GB
- PCo 15.1 SP03
- Log level set to "Error"
- Cache Mode Access to Cache, to Data Source as Required
- Agent instance running as Windows service
- Agent instance without subscription items Data source:
- Tag-based OPC UA server
- No security policies for the TCP/IP connection
- OPC UA server and PCo installed on the same Windows machine Network:
- SAP company network

The tests were performed using a Web socket test UI that read and write 1, 10, and 100 tags from and to PCo every 50 milliseconds. Every measurement was repeated 10 times to calculate an average. The following tables show the results of the measurements:

Running agent instance without any load put on it ("idle mode"):

Average CPU load	< 0.1%
Average memory consumption	53.0 MB

CPU load and memory consumption in idle mode depend on the source system type. In this example, the OPC UA source system monitors and keeps the session between the OPC UA server and PCo alive, which in total does not consume a lot of system performance.

The following tables show the measurement results for reading and writing 1, 10, and 100 tags at the same time at the highest frequency possible. Although the total time for reading and writing more than one tag is higher than for a single tag only, the results show that it is more efficient - from a per-tag perspective - to mass-read and write multiple data tags at the same time.

The examples are intended to give you an idea of the read and write frequency that PCo and the Web socket UI could handle.

**Read:**

	1 Tag	10 Tags	100 Tags
Time per read operation	10 ms	12 ms	43 ms
Time per read operation per tag	10 ms	1.2 ms	0.43 ms
Average CPU load	0.64 %	0.77 %	2.35 %
Peak CPU load	4.56 %	7.19 %	7.53 %
Average memory consumption	57.5 MB	57.7 MB	60.7 MB

**Write:**

	1 Tag	10 Tags	100 Tags
Time per write operation	10 ms	11 ms	41.0 ms
Time per write operation per tag	10 ms	0.11 ms	0.41 ms
Average CPU load	0.71 %	0.83 %	4.7 %
Peak CPU load	6.0 %	5.5 %	10.9 %
Average memory consumption	55.1 MB	55.2 MB	57.0 MB



## 6 WINDOWS PERFORMANCE MONITOR AND SAP PLANT CONNECTIVITY PERFORMANCE COUNTERS


You can measure the performance of a PCo connectivity scenario using the Windows standard tool *Performance Monitor*. PCo offers a set of performance counters that can be used from within the Performance Monitor.

To use the Windows Performance Monitor together with the PCo performance counters, proceed as follows:

1. You need to activate the performance counters for every agent instance. Stop the agent instance. Open a Windows Explorer window and navigate to the folder where the PCo agent instances configuration files are located (this is usually `C:\ProgramData\SAP\PCo\Config\Instances`). Open the `.exe-config` file of the agent instance in a text editor. Search for the line  

```
<add key="PerformanceCountersEnabled" value="false" />
```

Replace `false` with `true`, and then save the file.

2. Start the agent instance.
3. Start the Windows Performance Monitor by entering `perfmon` in the Windows search field, and choose .
4. In the Windows Performance Monitor, go to *Performance > Monitoring Tools > Performance Monitor*. Press the *Add* icon.
5. In the *Available Counters* list, you find a node *SAP Plant Connectivity* with all the performance counters provided by PCo. After selecting some or all of the performance counters, choose the PCo agent instance from the *Instances of selected objects* list. Then press *Add* and close the *Add counters* dialog.
6. If you want to measure the processor time that is occupied by the PCo agent instance, you choose the performance counter *Process > % ProcessorTime*. Choose the service *PCoSvcHost* from the *Instances of selected objects* list. Note: If you want the service *PCoSvcHost* to appear multiple times in the list, you need to stop other PCo agent instances and the PCo Management Host service to identify the service instance for the PCo agent instance that you want to measure.

You should disable the performance counters after the measurements by editing the `.exe-config` file of the agent instance again. Otherwise, in a high-performance scenario, the performance counters themselves could have a negative impact on the overall performance.

## 7 COMMENTS AND FEEDBACK

Comments and feedback are welcome. Please create a customer message on the SAP Support Portal with the following component: MFG-PCO.

[www.sap.com/contactsap](http://www.sap.com/contactsap)

© 2020 SAP SE. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company.

The information contained herein may be changed without prior notice. Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

In particular, SAP SE or its affiliated companies have no obligation to pursue any course of business outlined in this document or any related presentation, or to develop or release any functionality mentioned therein. This document, or any related presentation, and SAP SE's or its affiliated companies' strategy and possible future developments, products, and/or platform directions and functionality are all subject to change and may be changed by SAP SE or its affiliated companies at any time for any reason without notice. The information in this document is not a commitment, promise, or legal obligation to deliver any material, code, or functionality. All forward-looking statements are subject to various risks and uncertainties that could cause actual results to differ materially from expectations. Readers are cautioned not to place undue reliance on these forward-looking statements, and they should not be relied upon in making purchasing decisions.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies. See [www.sap.com/copyright](http://www.sap.com/copyright) for additional trademark information and notices.

**THE BEST RUN**

