D:Wave
The Quantum Computing Company™

Measuring Computation Time on D-Wave Systems

USER MANUAL

2018-11-27

Overview

This document describes the computation process, in the context of system timing, on D-Wave quantum computers. It explains the overall service time that is allocated to a problem, describes how the QPU is accessed within that period, and lists the timing-related fields that are available through the clients, including those that allow for user control.

CONTACT

Corporate Headquarters
3033 Beta Ave
Burnaby, BC V5G 4M9
Canada
Tel. 604-630-1428

US Office
2650 E Bayshore Rd
Palo Alto, CA 94303

Email: info@dwavesys.com

www.dwavesys.com

D-Wave User Manual 09-1107A-K

Notice and Disclaimer

D-Wave Systems Inc. (D-Wave), its subsidiaries and affiliates, makes commercially reasonable efforts to ensure that the information in this document is accurate and up to date, but errors may occur. NONE OF D-WAVE SYSTEMS INC., its subsidiaries and affiliates, OR ANY OF ITS RESPECTIVE DIRECTORS, EMPLOYEES, AGENTS, OR OTHER REPRESENTATIVES WILL BE LIABLE FOR DAMAGES, CLAIMS, EXPENSES OR OTHER COSTS (INCLUDING WITHOUT LIMITATION LEGAL FEES) ARISING OUT OF OR IN CONNECTION WITH THE USE OF THIS DOCUMENT OR ANY INFORMATION CONTAINED OR REFERRED TO IN IT. THIS IS A COMPREHENSIVE LIMITATION OF LIABILITY THAT APPLIES TO ALL DAMAGES OF ANY KIND, INCLUDING (WITHOUT LIMITATION) COMPENSATORY, DIRECT, INDIRECT, EXEMPLARY, PUNITIVE AND CONSEQUENTIAL DAMAGES, LOSS OF PROGRAMS OR DATA, INCOME OR PROFIT, LOSS OR DAMAGE TO PROPERTY, AND CLAIMS OF THIRD PARTIES.

D-Wave reserves the right to alter this document and other referenced documents without notice from time to time and at its sole discretion. D-Wave reserves its intellectual property rights in and to this document and its proprietary technology, including copyright, trademark rights, industrial design rights, and patent rights. D-Wave trademarks used herein include D-Wave®, D-Wave 2X™, D-Wave 2000Q™, Leap™, and the D-Wave logos (the D-Wave Marks). Other marks used in this document are the property of their respective owners. D-Wave does not grant any license, assignment, or other grant of interest in or to the copyright of this document, the D-Wave Marks, any other marks used in this document, or any other intellectual property rights used or referred to herein, except as D-Wave may expressly provide in a written agreement. This document may refer to other documents, including documents subject to the rights of third parties. Nothing in this document constitutes a grant by D-Wave of any license, assignment, or any other interest in the copyright or other intellectual property rights of such other documents. Any use of such other documents is subject to the rights of D-Wave and/or any applicable third parties in those documents.

All installation, service, support, and maintenance of and for the D-Wave System must be performed by qualified factory-trained D-Wave personnel. Do not move, repair, alter, modify, or change the D-Wave System. If the equipment is used in a manner not specified by D-Wave, the protection provided by the equipment may be impaired. Do not provide access to the customer site to anyone other than authorized and qualified personnel. Failure to follow these guidelines may result in disruption of service, extended downtime, damage to equipment (customer's, D-Wave's, and/or third parties'), injury, loss of life, or loss of property.

Contents

D-Wave User Manual 09-1107A-K

CHAPTER 1

ABOUT THIS DOCUMENT

1.1 Intended Audience

This document is for users of a D-Wave™ quantum computer system who want to understand how their use of the system is timed, which bears on how their use of the system will be accounted.

The document is written with increasing levels of detail in later sections. End users wishing only to understand how their use of the D-Wave system is being accounted can read only through Chapter 2. Application and tool developers and those assessing system performance will often want the detail of later sections.

1.2 Scope

This document describes the computation process, focusing on system timing, on D-Wave quantum computers. It explains the overall time that is allocated to a quantum machine instruction (QMI), describes how use of the quantum processing unit (QPU) is timed within that period, gives context for how timing can vary, and describes the timing-related fields in the Solver API (SAPI), both for user control and measurement.

1.3 Glossary

This document uses the following terms:

- Quantum processing unit (QPU): Quantum computational element within a D-Wave system.
- Quantum machine instruction (QMI): Set of information that is sent to the QPU, including the Ising model or QUBO parameters and annealing-cycle parameters.
- Annealing cycle: Physical process of starting with the QMI (prepared for the D-Wave system) and yielding a single sample to the QMI. Typically executed multiple times in a single QMI.
- Sample, result, read, or solution: Terms for the result yielded by an annealing cycle; 'sample' is preferred to connote the nondeterministic behavior of the QPU.

CHAPTER 2

TIMING OVERVIEW

Fig. 2.1 shows a simplified diagram of the sequence of steps, the dark red set of arrows, to execute a quantum machine instruction (QMI) on a D-Wave system, starting and ending on a user's client system. Each QMI consists of a single input together with parameters. A QMI is sent across a network to the SAPI server and joins a queue. Each queued QMI is assigned to one of possibly multiple workers, which may run in parallel. A worker prepares the QMI for the quantum processing unit (QPU) and optionally for postprocessing, sends the QMI to the QPU queue, receives samples (results) and optionally post-processes them (overlapping in time with QPU execution), and bundles the samples with additional QMI-execution information for return to the client system.

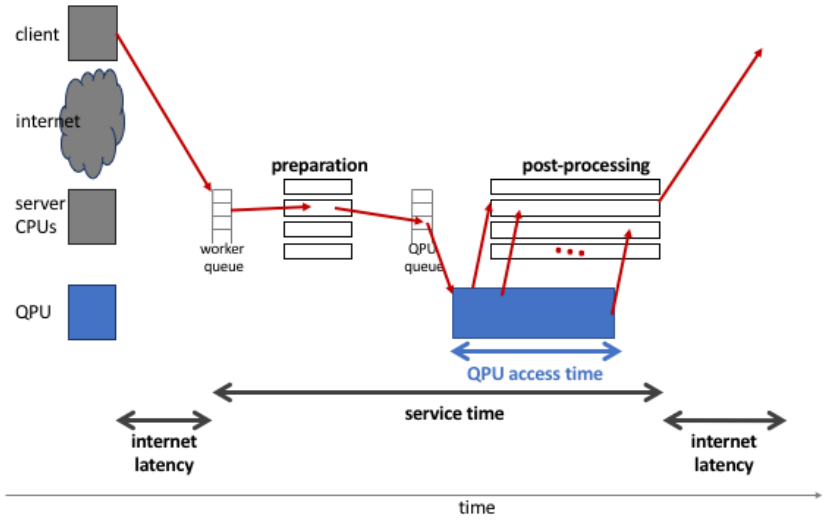

Figure 2.1: Overview of execution of a single QMI, starting from a client system, and distinguishing classical (client, CPU) and quantum (QPU) execution.

The total time for a QMI to pass through the D-Wave system is the *service* time. The execution time for a QMI as observed by a client includes service time and internet latency. The QPU executes one QMI at a time, during which the QPU is unavailable to any other QMI. This execution time is known as the QMI's *QPU access* time.

Important: Users executing on a D-Wave system are charged for QPU access time.

For customers purchasing time from D-Wave, this will be a financial charge. For users on a customer-owned system, this will typically be an administrative charge.

CHAPTER 3

BREAKDOWN OF QPU ACCESS TIME

As illustrated in Figure 3.1, the time to execute a single QMI on a QPU, *QPU access* time, is broken into two parts: a one-time initialization step to program the QPU (blue), and typically multiple (R) per-sample times for the actual execution on the QPU (repeated multicolor).

Note: A small amount of initialization time spent in low-level operations, denoted as Δ in the equations below, is roughly 1 ms and is reported by the Solver API *qpu_access_overhead_time* parameter.

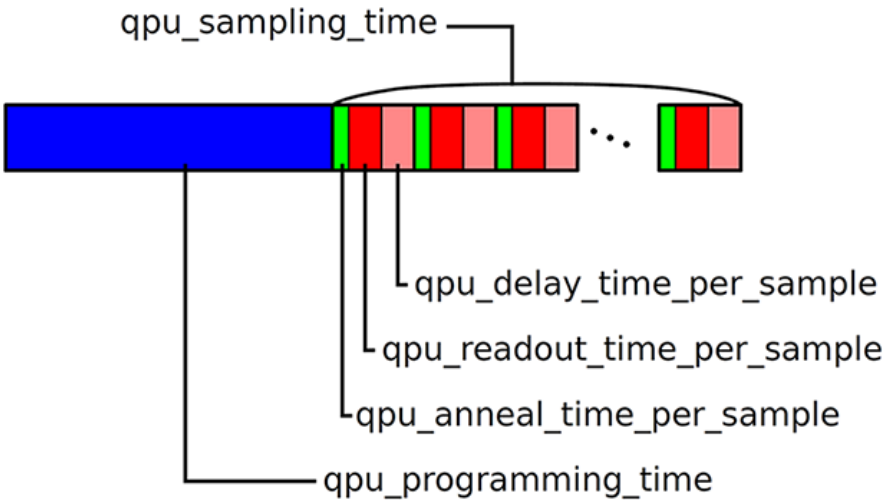

Figure 3.1: Detail of QPU access time.

The end-to-end time for one sample is further broken into anneal (the anneal proper; green), readout (read the sample from the QPU; red), and thermalization (wait for the QPU to regain its initial temperature; pink). Note that these component times are calculated as average times per sample. Possible rounding errors mean that the sum of these times may not match the total sampling time reported. Thus, for total time T, these times relate as follows:

$$\begin{aligned} T &= T_p + \Delta + T_s \\ T_s/R &\approx T_a + T_r + T_d, \end{aligned}$$

where

$$\begin{aligned} T_p &= \textit{programming\_time} \\ T_s &= \textit{sampling\_time} \end{aligned} \tag{3.1}$$

and

$$\begin{aligned} T_a = &\quad \text{(total) } \textit{annealing\_time} \\ T_r = &\quad \text{(total) } \textit{readout\_time} \\ T_d = &\quad \text{(total) } \textit{anneal\_schedule} \\ &+ \quad \text{(total) } \textit{readout\_thermalization} \\ &+ \quad \text{(total) } \textit{reduce\_intersample\_correlation} \\ &+ \quad \text{(total) } \textit{reinitialize\_state}. \end{aligned} \tag{3.2}$$

Note that *reinitialize_state* is used only for reverse annealing.

CHAPTER 4

SOURCES OF TIMING VARIATION AND ERROR

Running a D-Wave-using program across the internet or even examining QPU timing information may show variation from run to run from the end-user's point of view. This section describes some of the possible sources of such variation.

4.1 Nondedicated QPU Use

D-Wave systems are typically shared among multiple users, each of whom submits QMIs to solve a problem, with little to no synchronization among users. (A single user may also have multiple client programs submitting unsynchronized QMIs to a D-Wave system.) The QPU must be used by a single QMI at a time, so the D-Wave system software ensures that multiple QMIs flow through the system and use the QPU sequentially. In general, this means that a QMI may get queued for the QPU or some other resource, injecting indeterminacy into the timing of execution.

Note: Contact your D-Wave system administrator or D-Wave Support if you need to ensure a quiet system.

4.2 Nondeterminacy of Classical System Timings

Even when a system is quiet except for the program to be measured, timings often vary. As illustrated in Fig. 4.1, running a given code block repeatedly can yield different runtimes on a classical system, even though the instruction execution sequence does not change. Runtime distributions with occasional large outliers, as seen here, are not unusual.

Timing variations are routine, caused by noise from the operating system (e.g., scheduling, memory management, and power management) and the runtime environment (e.g., garbage collection, just-in-time compilation, and thread migration). [1] In addition, the internal architecture of the classical portion of the D-Wave system includes multiple hardware nodes and software servers, introducing communication among these servers as another source of variation.

[1] A more common practice in computational research is to report an alternative measurement called CPU time, which is intended to filter out operating system noise. However, CPU timers are only accurate to tens of milliseconds, and CPU times are not available for QPU time measurements. For consistency, we use wall clock times throughout.

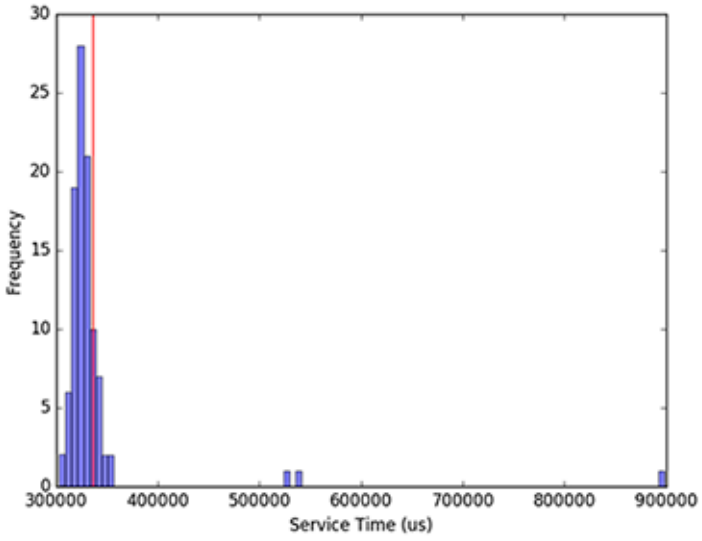

Figure 4.1: Histogram of 100 measurements of classical execution time using a wall clock timer, showing that the mean time of 336.5 ms (red line) is higher than 75 percent of the measurements.

For these reasons, mean reported runtimes can often be higher than median runtimes: for example, in Fig. 4.1, the mean time of 336.5 ms (vertical red line) is higher than 75 percent of the measured runtimes due to a few extreme outliers (one about 3 times higher and two almost 2 times higher than median). As a result, mean runtimes tend to exceed median runtimes. In this context, the smallest time recorded for a single process is considered the most accurate, because noise from outside sources can only increase elapsed time.[2] Because system activity increases with the number of active QMIs, the most accurate times for a single process are obtained by measuring on an otherwise quiet system.

Note: The 336 ms mean time shown for this particular QMI is not intended to be representative of QMI execution times.

The cost of reading a system timer may impose additional measurement errors, since querying the system clock can take microseconds. To reduce the impact of timing code itself, a given code block may be measured outside a loop that executes it many times, with running time calculated as the average time per iteration. Because of system and runtime noise and timer latency, component times measured one way may not add up to total times measured another way.[3] These sources of timer variation or error are present on all computer systems, including the classical portion of D-Wave platforms. Normal timer variation as described here may occasionally yield atypical and imprecise results; also, one expects wall clock times to vary with the particular system configuration and with system load.

[2] Randal E. Bryant and David R. O'Hallaron, *Computer Systems: A Programmer's Perspective (2nd Edition)*, Pearson, 2010.

[3] Paulo Eduardo Nogueira, Rivalino Matias, Jr., and Elder Vicente, *An Experimental Study on Execution Time Variation in Computer Experiments*, ACM Symposium on Applied Computing, 2014.

4.3 Internet Latency

If you are running your program on a client system geographically remote from the D-Wave system on which you're executing, you will likely encounter latency and variability from the internet connection itself (see Fig. 2.1).

4.4 Settings of User-Specified Parameters

The following user-specified parameters can cause timing to change, but should not affect the variability of timing. For more information on these parameters, see *Solver Properties and Parameters Reference*.

- *anneal_schedule*—User-provided anneal schedule. Specifies the points at which to change the default schedule. Each point is a pair of values representing time t in microseconds and normalized anneal fraction s. The system connects the points with piecewise-linear ramps to construct the new schedule. If *anneal_schedule* is specified, T_a, *qpu_anneal_time_per_sample* is populated with the total time specified by the piecewise-linear schedule. Cannot be used with the *annealing_time* parameter.
- *annealing_time*—Duration, in microseconds, of quantum annealing time. This value populates T_a, *qpu_anneal_time_per_sample*. Cannot be used with the *anneal_schedule* parameter.
- *num_reads*—Number of samples to read from the solver per QMI.
- *num_spin_reversal_transforms*—For QMIs with more than one spin-reversal transform, SAPI handles the timing information for all the subQMIs that it sends to the solver as follows: (1) It sums each timing field that does not end with "per_sample." (2) For others, it sends the value from the first subQMI. For example, the values for *qpu_access_time* are summed; those from *qpu_delay_time_per_sample* are not.
- *programming_thermalization*—Number of microseconds to wait after programming the QPU to allow it to cool; i.e., *post-programming thermalization time*. Values lower than the default accelerate solving at the expense of solution quality. This value contributes to the total T_p, *qpu_programming_time*.
- *readout_thermalization*—Number of microseconds to wait after each sample is read from the QPU to allow it to cool to base temperature; i.e., *post-readout thermalization time*. This optional value contributes to T_d, *qpu_delay_time_per_sample*.
- *reduce_intersample_correlation*—Used to reduce sample-to-sample correlations. When true, adds to T_d, *qpu_delay_time_per_sample*. Amount of time added increases linearly with increasing length of the anneal schedule.
- *reinitialize_state*—Used in reverse annealing. When `true` (the default setting), reinitializes the initial qubit state for every anneal-readout cycle, adding between 100 and 600 microseconds to T_d, *qpu_delay_time_per_sample*. When `false`, adds approximately 10 microseconds to T_d.[4]
- *postprocess*—Specifies the type of (classical) postprocessing to be performed on the raw samples from the QPU. Requesting no postprocessing consumes the least time;

[4] Amount of time varies by system.

either sampling or optimization postprocessing consumes more.

Note: Depending on the parameters chosen for a QMI, QPU access time may be a large or small fraction of service time. E.g., a QMI requesting a single sample with short *annealing_time* would see programming time as a large fraction of service time and QPU access time as a small fraction.

Note: The *readout_thermalization* parameter is deprecated and will eventually be removed from the API. Plan code updates accordingly.

4.5 Upper Limit on User-Specified Timing-Related Parameters

The D-Wave system limits your ability to submit a long-running QMI to prevent you from inadvertently monopolizing QPU time. This limit varies by system; check the *problem_run_duration_range* property for your solver.

The limit is calculated according to the following formula:

$$Duration = ((P_1 + P_2) * P_3) + P_4 \tag{4.1}$$

where P_1, P_2, P_3, and P_4 are the values specified for the *annealing_time*, *readout_thermalization*, *num_reads* (samples), and *programming_thermalization* parameters, respectively.

If you attempt to submit a QMI whose execution time would exceed the limit for your system, an error is returned showing the values in microseconds. For example:

```
ERROR: Upper limit on user-specified timing related parameters exceeded: 12010000 >␣
↪3000000
```

Note that it is possible to specify values that fall within the permitted ranges for each individual parameter, yet together cause the time to execute the QMI to surpass the limit.

CHAPTER 5

BREAKDOWN OF SERVICE TIME

The service time can be broken into:

- Any time required by the worker before and after QPU access
- Wait time in queues before and after QPU access
- QPU access time
- Postprocessing (PP) time

Service time is defined as the difference between time-in and time-out for each QMI, as shown in the table.

Keyword in SAPI	Meaning
time_received	When QMI arrives
time_solved	When bundled samples are available

As mentioned in the introduction, service time for a single QMI depends on the system load; that is, how many other QMIs are present at a given time. During periods of heavy load, wait time in the two queues may contribute to increased service times. D-Wave has no control over system load under normal operating conditions. Therefore, it is not possible to guarantee that service time targets can be met. Service time measurements described in other D-Wave documents are intended only to give a rough idea of the range of experience that might be found under varying conditions.

5.1 Postprocessing Time

You can apply one or more postprocessing utilities to the raw samples returned by the QPU. As shown in Fig. 5.1, postprocessing (red) works in parallel with sampling (blue), so that the computation times overlap except for postprocessing the last batch of samples. In this diagram, the time consumed by gathering small batches of samples are marked by vertical blue lines. Within execution of a QMI, gathering the current set of samples takes place concurrently with postprocessing for the previous set of samples (red boxes), which is applied to batches of samples as they are returned by the QPU. As illustrated by Fig. 5.1, only the time for postprocessing the last set of samples (the rightmost red box) is not overlapped with sampling.

Postprocessing overhead is designed not to impose any delay to QPU access for the next QMI, because postprocessing of the last batch takes place concurrently with the next QMI's programming time.

The system returns two associated timing values, as shown in the table below. Referring to Fig. 5.1, *total_post_processing_time* is the sum of all times in the red boxes, while

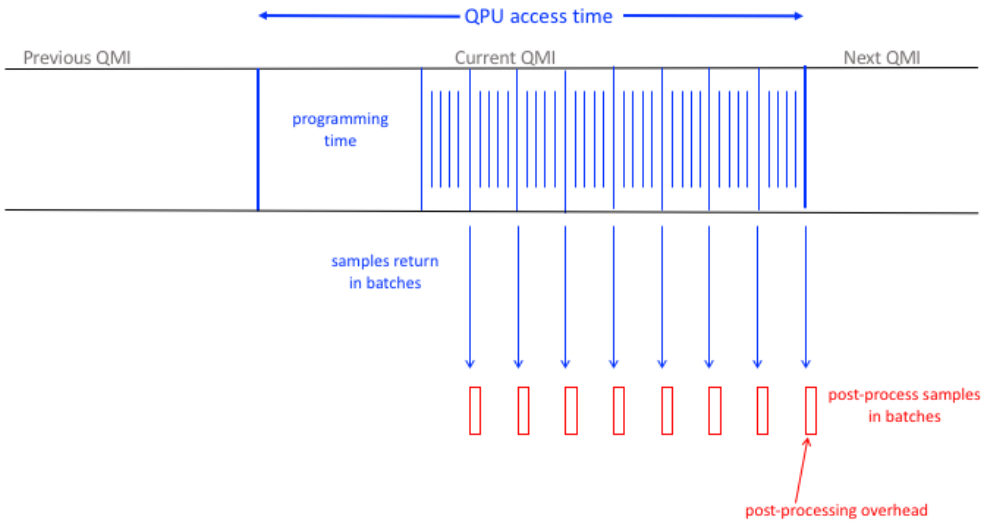

Figure 5.1: Relationship of QPU time to postprocessing time, illustrated by one QMI in a sequence (previous, current, next).

post_processing_overhead is the extra time needed (a single red box) to process the last batch. This latter time together with *qpu_access_time* contributes to overall service time.

Note: Even if no postprocessing is run on a QMI, the returned *post_processing_overhead* value is non-zero. This is because computing the final energies occurs after samples are returned and is accounted as postprocessing overhead.

Keyword in SAPI	Meaning
total_post_processing_time	Total time for postprocessing
post_processing_overhead_time	Added for the last batch

For more details about postprocessing and how it is handled in the timing structure, see *Postprocessing Methods on D-Wave Systems*.

5.2 "Total Time" Reported in Statistics (for Administrators)

One timing parameter, *qpu_access_time*, provides the raw data for the "Total Time" values reported as system statistics, available to administrators. Reported statistics are the sum of the *qpu_access_time* values for each QMI selected by the users, solvers, and time periods selected in the filter.

Note: Reported statistics are in milliseconds, while SAPI inputs and ouputs are in mi-

croseconds. One millisecond is 1000 microseconds.

CHAPTER 6

USING TIMING INFORMATION: DWAVE-CLOUD-CLIENT

6.1 Timing-Related Fields

The table below lists the timing-related fields available through the dwave-cloud-client Python package. The *timing* object is a field in the data structure returned by the dwave-cloud-client *sample_ * and *solve_ * methods. The fields such as *qpu_access_time* can be indexed in *timing* as ['*timing*']['*qpu_access_time*'].

Table 6.1: Fields that affect *qpu_access_time*

Value	Field in dwave-cloud-client	Meaning	Affected by
T	*qpu_access_time*	Total time in QPU	All parameters listed below
T_p	*qpu_programming_time*	Total time to program the QPU[1]	*programming_thermalization*
Δ		Additional low-level operations	
R		Number of reads (samples)	
T_s	*qpu_sampling_time*	Total time for R samples	*num_reads*
T_a	*qpu_anneal_time_per_sample*	Time for one anneal	*anneal_schedule*, *annealing_time*, *num_spin_reversal_transforms*
T_r	*qpu_readout_time_per_sample*	Time for one read	*num_spin_reversal_transforms*
T_d	*qpu_delay_time_per_sample*	Delay between anneals[2]	*anneal_schedule*, *num_spin_reversal_transforms*, *readout_thermalization*, *reduce_intersample_correlation*, (only in case of reverse annealing) *reinitialize_state*

[1] Even if *programming_thermalization* is 0, T_p is several milliseconds for other operations.

[2] The time returned in the *qpu_delay_time_per_sample* field is equal to a constant plus the user-specified value, *readout_thermalization*.

6.2 Timing Data Returned by the dwave-cloud-client

Below is a sample skeleton of Python code for accessing timing data returned by dwave-cloud-client. Timing values are returned in the computation object and the timing object; further code could query those objects in more detail. The timing object referenced on line 16 is a Python dictionary containing (key, value) pairs. The keys match keywords discussed in this section.

```
01 import random
02 import datetime as dt
03 from dwave.cloud import Client

04 # Connect using the default or environment connection information
05 with Client.from_config() as client:

06     # Load the default solver
07     solver = client.get_solver()

08     # Build a random Ising model to exactly fit the graph the solver supports
09     linear = {index: random.choice([-1, 1]) for index in solver.nodes}
10     quad = {key: random.choice([-1, 1]) for key in solver.undirected_edges}

11     # Send the problem for sampling, include solver-specific parameter 'num_reads
 ↪'
12     computation = solver.sample_ising(linear, quad, num_reads=100)
13     computation.wait()

14     # Print the first sample out of a hundred
15     print(computation.samples[0])
16     timing = computation['timing']


17     # Service time
18     time_format = "%Y-%m-%d %H:%M:%S.%f"
19     start_time = dt.datetime.strptime(str(computation.time_received)[:-6], time_
 ↪format)
20     finish_time = dt.datetime.strptime(str(computation.time_solved)[:-6], time_
 ↪format)
21     service_time = finish_time - start_time
22     qpu_access_time = timing['qpu_access_time']
23     print("start_time="+str(start_time)+", finish_time="+str(finish_time)+ \
24             ", service_time="+str(service_time)+", qpu_access_time="       \
25             +str(float(qpu_access_time)/1000000))
```

CHAPTER 7

USING TIMING INFORMATION: SAPI CLIENTS

Note: This section is relevant to users of the C, MATLAB, and Python client libraries. If you are using the dwave-cloud-client in the Ocean SDK instead, see *Using Timing Information: dwave-cloud-client*.

7.1 Timing-Related Fields

The table below lists the timing-related fields available through the SAPI client libraries.

Table 7.1: Fields that affect *qpu_access_time*

Value	Field in SAPI	Meaning	Affected by
T	*qpu_access_time*	Total time in QPU	All parameters listed below
T_p	*qpu_programming_time*	Total time to program the QPU[1]	*programming_thermalization*
Δ		Additional low-level operations	
R		Number of reads (samples)	
T_s	*qpu_sampling_time*	Total time for R samples	*num_reads*
T_a	*qpu_anneal_time_per_sample*	Time for one anneal	*anneal_schedule*, *annealing_time*, *num_spin_reversal_transforms*
T_r	*qpu_readout_time_per_sample*	Time for one read	*num_spin_reversal_transforms*
T_d	*qpu_delay_time_per_sample*	Delay between anneals[2]	*anneal_schedule*, *num_spin_reversal_transforms*, *readout_thermalization*, *reduce_intersample_correlation*, (only in case of reverse annealing) *reinitialize_state*

[1] Even if *programming_thermalization* is 0, T_p is several milliseconds for other operations.

[2] The time returned in the *qpu_delay_time_per_sample* field is equal to a constant plus the user-specified value, *readout_thermalization*.

7.2 Timing Data Returned by SAPI Clients

Below is a sample skeleton of Python code for accessing timing data returned by SAPI. Timing values are returned in the status object and the timing object; further code could query those objects in more detail. The timing object referenced on line 10 is a Python dictionary containing (key, value) pairs. The keys match keywords discussed in this section.

```
01 from dwave_sapi2.local import local_connection
02 from dwave_sapi2.core import async_solve_ising, await_completion
03 import datetime

04 h,J = build_hamiltonian()
05 solver = local_connection.get_solver('example_solver')
06 submitted_qmi = async_solve_ising(solver, h, J, num_reads=100)
07 await_completion([submitted_qmi], min_done=2, timeout=1.0)

08 # QPU and PP times
09 result = submitted_qmi.result()
10 timing = result['timing']

11 # Service time
12 time_format = "%Y-%m-%dT%H:%M:%S.%fZ"
13 status = submitted_qmi.status()
14 start_time = datetime.datetime.strptime(status ['time_received'], time_format)
15 finish_time = datetime.datetime.strptime(status['time_solved'], time_format)
16 service_time = finish_time - start_time
```