Development of Autograder For Competitive Programming Using Contestant PC As Worker

Jauhar Arifin
School of Electrical Engineering and Informatics

Bandung Institute of Technology
Bandung, Indonesia
jauhararifin10@gmail.com

Riza Satria Perdana
School of Electrical Engineering and Informatics

Bandung Institute of Technology
Bandung, Indonesia
riza@informatika.org

Abstract—**Competitive programming is a computer science competition where the contestants compete to solve computer science problems by writing a program which satisfies the problem constraints. Autograder is used to grade contestant solutions automatically in real-time. Usually autograder is deployed in many computers to increase grading performance. In this work, contestant computers are used as worker to run autograder. By using contestant computers as worker, the number of worker will proportional to the number of contestant submissions, thus increasing grading performance. Every contestant computers have different specification and can affect grading fairness. To keep grading fairness, contestant's solution and jury's solution executed in contestant worker and compared to check whether contestant's solution satisfies problem constraints. This work tested by simulating grading process in contestant computers. The testing result indicates that using contestant computers as worker gives performance improvement in the grading process.**

Index Terms—**competitive programming, online judge, autograder.**

I. Introduction

Competitive programming is one of the most popular competition in computer science field. In competitive programming competition, contestants are asked to solve computer science problems correctly and as fast as possible. Some institute and organization often organize competitive programming competition periodically. Some big companies like Google and Facebook organize competitive programming competition annually. Competitive programming competition supported by online judge system. Usually, online judge system is a web based application where contestants can read the problems, create clarifications, submit their solution, and watch the scoreboard. Currently, the most popular online judge are Codeforces, URI Online Judge [8], Uva, and SPOJ.

In order to grade contestant submissions, online judge system has a subsystem called autograder. Contestant submissions which are source code in certain programming language will be graded by autograding system by compiling the program and executes compiled program using test-cases that have been prepared by juries or problem setter. According [14], this grading method is called black-box grading. By using autograding system, the grading process can be done automatically and juries don't have to evaluate contestant submissions manually. In order to increase the number of contestant submissions in certain amount of time, juries usually deploy autograder in many computers. In order to run autograder in many computers, the juries need to prepare many computers with the same spesification to keep the fairness of grading process.

Currently, almost every competitive programing competition use online judge system to support the competition and use many computers to run autograder in order to improve grading performance. Every computers that run autograder are called worker. Grading performance is defined as the number of submissions graded in certain amount of time. Even though autograder deployed in many computers, the grading performance often not enough to evaluate contestant solutions in real-time because the number of contestants is increasing. Furthermore, the number of computer to deploy autograder affect the procurement cost that juries need to be incurred. Therefore, new grading system is needed to increase grading performance without increasing procurement cost that need to be incurred.

When competing in competitive programming competition, the contestants usually use their personal computer to write the solutions. Every contestant computers usually have sufficient specification to compile and execute contestant submissions. Therefore, contestant computers have ability to run autograder program and evaluate contestant submissions.

II. Related Work

There are some popular online judge system that have been used to organize competitive programming competition. Some online judge offer another additional features like discussion forum, training gate and rating system. Nowadays, most of online judge deployed as web application and use specific computer to grade contestant submissions. There are many types of competitive programming competition rules. Most of online judge only support a specific competition rules.

One of the most popular competitive programming competition is ACM-ICPC competition where the competition uses ICPC rules. In ACM-ICPC competition, contestants compete in group of three people. Every group have the same amount of problem to solve, and the score is determined by the number of solved problem and time penalty. There are many online judges that support this competitive programming rules. The most popular online judge that support this type of competition is DOMJudge. DOMJudge is very popular because it has many usefull feature, easy to use and open source.

Currently, most of online judge use autograder to evaluate contestant submissions. According to [11], autograder is a system that compiles, executes and evaluate source code. Evaluates source code manually takes three minutes while using autograder only need ten seconds. By using autograder, contestants can receive grading feedback faster and reduce the work that need to be done by juries. Basically, the autograder evaluate the source code by compiles the source code, and executes the compiled program using predefined test-cases. The output of contestant's solution program compared to the predefined test-case and then graded. This evaluation method is called black-box grading [14].

We need to consider security aspect when developing autograding system. Competitive programming contestants might submit source code that contain dangerous code. According to [9], there are several attacks that can be made by contestants, such as submitting compile bomb code, submitting code that destroy autograding environment, and submitting code that access forbidden computer's resource. In order to prevent such things, autograding system must execute contestant's code in isolated environment called sandbox.

There are several ways to create isolated environment such as using virtual machine and containerization. By using virtual machine, one can create isolated operating system inside host computer. Virtual machine uses hypervisor to emulate computer hardware and run its own operating system. In Linux based operating system, there is a feature called KVM that make it possible to create hypervisor as a process in Linux [10]. We can use virtual machine to isolate the execution of contestant's code. However, by using virtual machine, we need to boot a new operating system each time we evaluate contestant's subission. This booting process takes so much computer's resource and reduces grading performance.

Another way to create isolated environment is by using container. Container is different from virtual machine. Container Container gives isolation in software level wihout hypervisor [12]. By using container, we can isolate process execution without boot a new operating system. Container works by using some Linux features such as chroot, namespace and cgroup. By using chroot, we can isolate filesystem of Linux process [13]. Container uses Linux feature called cgroup to limit process resources. By using cgroup, we can limit memory usage, CPU usage and disk IO usage of running process [10]. In order to hide other process, user, and network from a running process, container uses a Linux feature called namespace. By using namespace, we can isolate a Linux process from other process, so the running process not aware of the existance of other process. Instead of virtual machine, most of autograding system use container to isolate grading process. This method is used because it is more lighter and faster than virtual machine.

III. Methodology

A. Time and Memory Measurement

In competitive programming competition, every problems has time and memory constraints. Contestants are asked to solve the problem by writing program that satify the constraints. According to ICPC rules, when contestant's solution exceeds the problem constraints, the solution is rejected and the contestant get time penalty. Thus, autograding system need to tell whether contestant's solution exceeds the constraints or not. The probem is, every computer has different performance when executing a program. The same program might run faster in one computer and slower in another computer. Currently, most of online judge system uses identical computer to evaluate contestant submissions. By using identical computer, the same program expected to run at the same duration and consume the same amount of memories.

Contestant computers have different specification and performance. We cannot force the contestants to have identical computer. Thus, we need new time and memory measurement method. The performance of a computer is determined by its specifications such as CPU clock speed and operating system. We can tell the computer performance by its specifications. However, besides CPU clock speed and operating system, there are so much other factors that affect computer performance such as temperature, cache size, the program itself, and many other factors. Thus, determining computer performance by its specification is very hard and not feasible to do.

$$R_{contestant} < (1+\gamma) \times R_{jury} \tag{1}$$

In this work, we compare contestant's solution and jury's solution. Contestant's solution considered satisfying the problem constraints when the memory and CPU usage of contestant's solution not exceeding jury's solution. Sometimes, contestant solution's memory or CPU exceeds jury's solution a little bit because the implementation is different. We add tolerance factor (γ) to tolerate this problem. Contestant's solution is considered satifying problem constraints when the Equation 1 is fulfilled. When evaluating contestant's solution, the worker compiles and executes contestant's solution and jury's solution. After executes the solutions, the worker compares the memory and CPU usage of the execution and determine whether the contestant's solution satisfy problem constraints.

B. Load Balancing

We need to distribute grading jobs to worker evenly in order to increase grading performance. There are several load balancing method that we can use to distribute grading jobs evenly. The easiest way to distribute the jobs is by making each contestants to grade their own submissions. In this work, we didn't use this method because this method creates security vulnerabilites.

The other way distribute grading jobs is by using push-based load balancing method. In push-based load balancing, the online judge server choose the worker to do the grading jobs. By using this method, every contestant doesn't know which submission they grade, thus can increase system security. To improve system security, we can also grade every submission multiple times. In order to implement this method, we need the status of all workers. This method is pretty hard to implement because we need to poll the worker status periodically and choose the best worker to evaluate the contestant submissions.

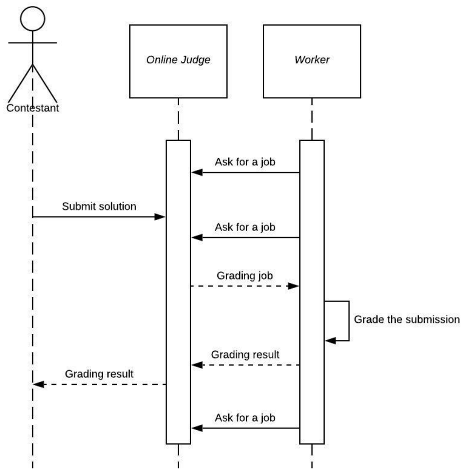

Fig. 1. Pull-based Load Balancing.

In this work, we use pull-based load balancing method. By using this method, we don't have to poll the worker status periodically. The worker will poll the jobs from server periodically and evaluate them when the there is a job that ready to be evaluated. Fig. 1 depicts this method mechanism. This work is easier to implements than push-based load balancing method and has the same advantages.

C. Evaluating Submission In Isolated Environment

In order to isolate the evaluation process, we use containerization method. We compile and execute contestant's submission inside lightweight container. This container is created by using several Linux features and system calls such as cgroup, namespace, chroot and setrlimit. We use cgroup to monitor and limit the CPU and memory consumption of running process. The worker kills the program when the CPU or memory usage is exceeding the limit. In order to isolate the host's user, process and network, we use Linux feature called namespace. By using namespace, the processes inside isolated environment will not aware of any process, user or network in the outside of its own environment. In order to isolate filesystem of a process, we use chroot system call. Beside isolating the filesystem, we also need to bring some files from host environent into the isolated environment. We bring some of host filesystem into the isolated environment by using mount system call. We also limit the process limit by using setrlimit system call. By using setrlimit system call, we limit some process resources such as the number of child processes, the stack size, the number of open files and the size of file created by that process. Fig. 2 depicts how isolated environment is created.

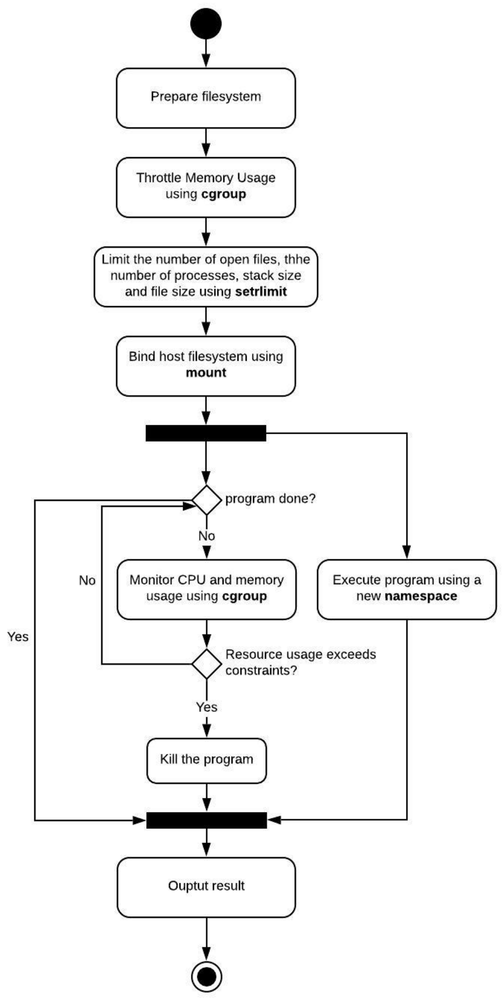
Prepare filesystem
Throttle Memory Usage using **cgroup**
Limit the number of open files, thhe number of processes, stack size and file size using **setrlimit**
Bind host filesystem using **mount**
program done?
No
Monitor CPU and memory usage using **cgroup**
Execute program using a new **namespace**
No
Yes
Resource usage exceeds constraints?
Yes
Kill the program
Ouptut result

Fig. 2. Sandbox Activity Diagram.

D. Program Compilation

In order to evaluate contestant's submissions, we need to compile and execute the submitted source code. To compile the source code, we need a specific compiler. Every worker should has the same version of compiler. The same compiler with different version can produces different output. In order to keep the fairness of grading process, we need to distribute the same version of compiler to all workers in the system. In this work, we distribute the compiler in the autograder program to every workers. Every worker compiles the source code inside an isolated filesystem that has been installed with the same version of compilers.

E. Obtaining Problem Test-cases

Every worker needs test-cases to evaluate contestant solutions. A test-case is a pair of text file that contains input and output of the problem. Usually, the test-case files are generated by test-case generator that created by problem setters. The worker evaluates the contestant solution by executing the solution using the input test-case and compare the output with the output test-case. In order to evaluate the contestant solutions, we need to bring the problem testcases into the worker. To bring the test-cases into the worker is not easy because the test-case file can be a very large file and confidential.

In this work, we don't transfer the raw test-case files to all the workers. Transfering raw test-case files needs a huge bandwidth and can reduce grading performance. Instead of transfering raw test-case file, we transfer the test-case generator program to generate test-case files. The worker receive test-case generator source code from the server and compiles it. The compiled test-case generator program then executed to generates input test-case. The worker uses jury's solution to generates output test-case files. The worker also needs checker program to compare contestant's output with jury's output. The checker program receive contestant's output and jury's output as its input and gives the verdict of the submission.

In order to hide the test-case information, we encrypt the test-case generator program and the generated test-case files. Even though the test-case is encrypted, the contestants might decrypt it if they know the encryption key. The contestant can attacks the autograder program using reverse engineering technique to obtain the encryption key. In this work, we didn't handle this problem. Further research are needed to solve this problem.

F. Handling Reverse Engineering Attack

Because the autograder programs are deployed in the contestant computers, the contestants can do everything to the program. The contestant can attacks the autograder program in their computer. There are many kind of attacks that the contestant can do such as changing the compiler program, sending the false verdict to the server and refuse to grade other contestant submission. In this work, we didn't handle this kind of attacks, but there is a way to handle this kind of attacks.

We can prevent a contestant to refuse do the grading process by monitoring the grading process. When we find out that there is a worker that not asks for job for a long time or not sends the grading result for a long time, we can tell that there is a contestant that attacks the autograder program in their computer. Then, we can warn or disquafy the contestant for attacking the autograder program.

By grading every submission multiple times, we can prevent a worker from being atacked that cause it gives wrong verdict after grading. When we grade a submission multiple times and there is a contestant that attacks the autograder in their computer, we can tell by investigating the grading result. If the grading result of a worker is different from the rest of workers, we can tell that the contestant attacks the autograder program in that worker. Then, we can warn or disquafy that contestant for attacking the autograder program.

There are many more type of attacks that we didn't handle in this work such as stealing jury's solution, other contestant's solution and test-case generator. Even though we didn't handle some type of attacks, this system can still works in several competition such as ACM-ICPC. In ACM-ICPC, every team is given a computer by the juries to solve the problems. The juries can install autograder program inside contestant's computer using root user and not giving the root access to the contestant. By using this mechanism, the contestant can't attacks the autograder program inside their computer.

G. Abbreviations and Acronyms

Define abbreviations and acronyms the first time they are used in the text, even after they have been defined in the abstract. Abbreviations such as IEEE, SI, MKS, CGS, ac, dc, and rms do not have to be defined. Do not use abbreviations in the title or heads unless they are unavoidable.

H. Units

- Use either SI (MKS) or CGS as primary units. (SI units are encouraged.) English units may be used as secondary units (in parentheses). An exception would be the use of English units as identifiers in trade, such as "3.5-inch disk drive".
- Avoid combining SI and CGS units, such as current in amperes and magnetic field in oersteds. This often leads to confusion because equations do not balance dimensionally. If you must use mixed units, clearly state the units for each quantity that you use in an equation.
- Do not mix complete spellings and abbreviations of units: "Wb/m^2" or "webers per square meter", not "webers/m^2". Spell out units when they appear in text: ". . . a few henries", not ". . . a few H".
- Use a zero before decimal points: "0.25", not ".25". Use "cm^3", not "cc".)

I. Equations

Number equations consecutively. To make your equations more compact, you may use the solidus (/), the exp function, or appropriate exponents. Italicize Roman symbols for quantities and variables, but not Greek symbols. Use a long dash rather than a hyphen for a minus sign. Punctuate equations with commas or periods when they are part of a sentence, as in:

$$a + b = \gamma \tag{2}$$

Be sure that the symbols in your equation have been defined before or immediately following the equation. Use "(2)", not "Eq. (2)" or "equation (2)", except at the beginning of a sentence: "Equation (2) is . . ."

J. LaTeX-Specific Advice

Please use "soft" (e.g., `\eqref{Eq}`) cross references instead of "hard" references (e.g., `(1)`). That will make it possible to combine sections, add equations, or change the order of figures or citations without having to go through the file line by line.

Please don't use the `{eqnarray}` equation environment. Use `{align}` or `{IEEEeqnarray}` instead. The `{eqnarray}` environment leaves unsightly spaces around relation symbols.

Please note that the `{subequations}` environment in LaTeX will increment the main equation counter even when there are no equation numbers displayed. If you forget that, you might write an article in which the equation numbers skip from (17) to (20), causing the copy editors to wonder if you've discovered a new method of counting.

BibTeX does not work by magic. It doesn't get the bibliographic data from thin air but from .bib files. If you use BibTeX to produce a bibliography you must send the .bib files.

LaTeX can't read your mind. If you assign the same label to a subsubsection and a table, you might find that Table I has been cross referenced as Table IV-B3.

LaTeX does not have precognitive abilities. If you put a `\label` command before the command that updates the counter it's supposed to be using, the label will pick up the last counter to be cross referenced instead. In particular, a `\label` command should not go before the caption of a figure or a table.

Do not use `\nonumber` inside the `{array}` environment. It will not stop equation numbers inside `{array}` (there won't be any anyway) and it might stop a wanted equation number in the surrounding equation.

K. Some Common Mistakes

- The word "data" is plural, not singular.
- The subscript for the permeability of vacuum μ_0, and other common scientific constants, is zero with subscript formatting, not a lowercase letter "o".
- In American English, commas, semicolons, periods, question and exclamation marks are located within quotation marks only when a complete thought or name is cited, such as a title or full quotation. When quotation marks are used, instead of a bold or italic typeface, to highlight a word or phrase, punctuation should appear outside of the quotation marks. A parenthetical phrase or statement at the end of a sentence is punctuated outside of the closing parenthesis (like this). (A parenthetical sentence is punctuated within the parentheses.)
- A graph within a graph is an "inset", not an "insert". The word alternatively is preferred to the word "alternately" (unless you really mean something that alternates).
- Do not use the word "essentially" to mean "approximately" or "effectively".
- In your paper title, if the words "that uses" can accurately replace the word "using", capitalize the "u"; if not, keep using lower-cased.
- Be aware of the different meanings of the homophones "affect" and "effect", "complement" and "compliment", "discreet" and "discrete", "principal" and "principle".
- Do not confuse "imply" and "infer".
- The prefix "non" is not a word; it should be joined to the word it modifies, usually without a hyphen.
- There is no period after the "et" in the Latin abbreviation "et al.".
- The abbreviation "i.e." means "that is", and the abbreviation "e.g." means "for example".

An excellent style manual for science writers is [21].

L. Authors and Affiliations

The class file is designed for, but not limited to, six authors. A minimum of one author is required for all conference articles. Author names should be listed starting from left to right and then moving down to the next line. This is the author sequence that will be used in future citations and by indexing services. Names should not be listed in columns nor group by affiliation. Please keep your affiliations as succinct as possible (for example, do not differentiate among departments of the same organization).

M. Identify the Headings

Headings, or heads, are organizational devices that guide the reader through your paper. There are two types: component heads and text heads.

Component heads identify the different components of your paper and are not topically subordinate to each other. Examples include Acknowledgments and References and, for these, the correct style to use is "Heading 5". Use "figure caption" for your Figure captions, and "table head" for your table title. Run-in heads, such as "Abstract", will require you to apply a style (in this case, italic) in addition to the style provided by the drop down menu to differentiate the head from the text.

Text heads organize the topics on a relational, hierarchical basis. For example, the paper title is the primary text head because all subsequent material relates and elaborates on this one topic. If there are two or more sub-topics, the next level head (uppercase Roman numerals) should be used and, conversely, if there are not at least two sub-topics, then no subheads should be introduced.

N. Figures and Tables

a) Positioning Figures and Tables: Place figures and tables at the top and bottom of columns. Avoid placing them in the middle of columns. Large figures and tables may span across both columns. Figure captions should be below the figures; table heads should appear above the tables. Insert figures and tables after they are cited in the text. Use the abbreviation "Fig. 3", even at the beginning of a sentence.

TABLE I
TABLE TYPE STYLES

Table Head	**Table Column Head**		
	Table column subhead	***Subhead***	***Subhead***
copy	More table copy[a]		

[a]Sample of a Table footnote.

Figure Labels: Use 8 point Times New Roman for Figure labels. Use words rather than symbols or abbreviations when writing Figure axis labels to avoid confusing the reader. As an example, write the quantity "Magnetization", or "Magnetization, M", not just "M". If including units in the label, present them within parentheses. Do not label axes only with units. In

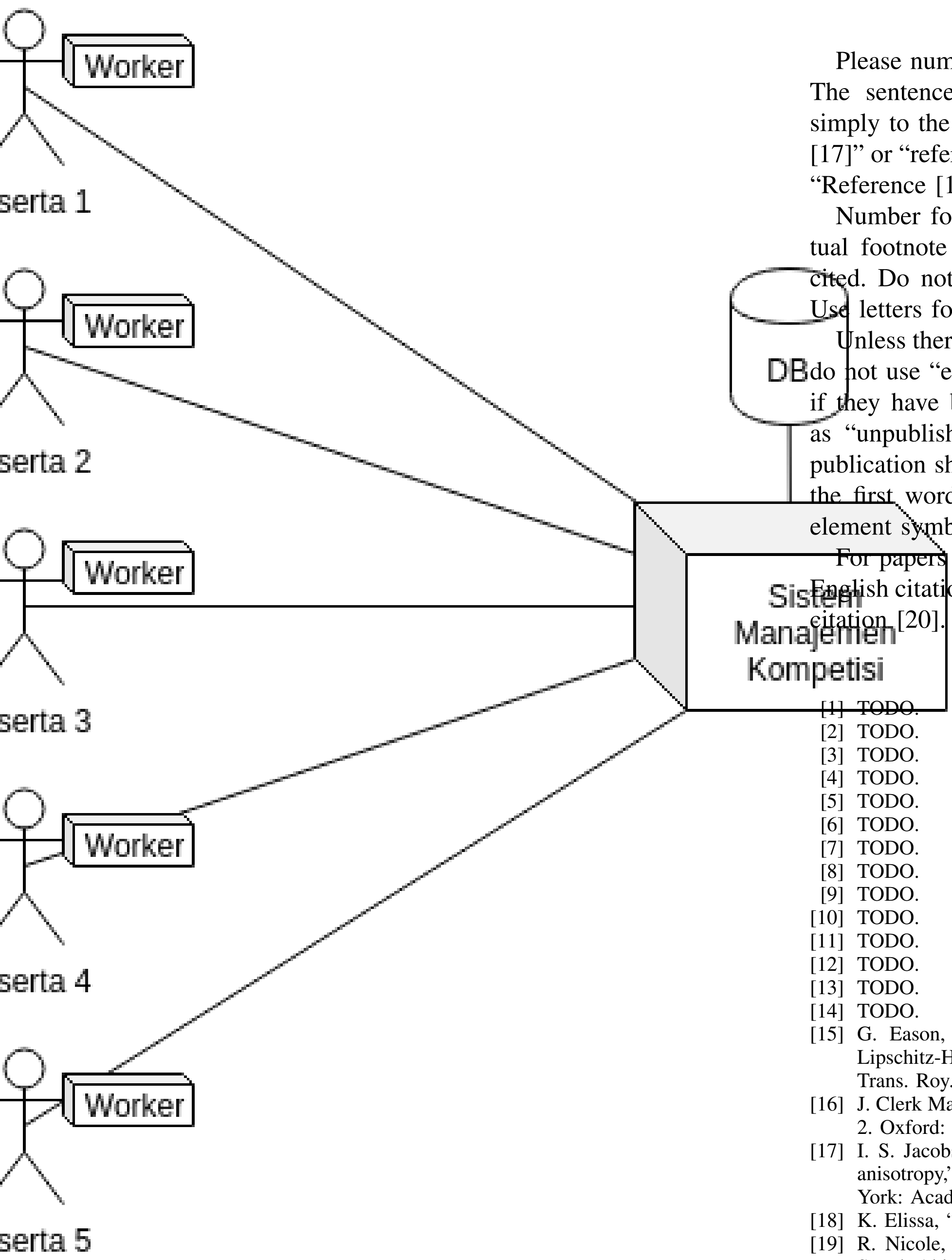

Fig. 3. Example of a figure caption.

the example, write "Magnetization (A/m)" or "Magnetization {A[m(1)]}", not just "A/m". Do not label axes with a ratio of quantities and units. For example, write "Temperature (K)", not "Temperature/K".

Acknowledgment

The preferred spelling of the word "acknowledgment" in America is without an "e" after the "g". Avoid the stilted expression "one of us (R. B. G.) thanks ...". Instead, try "R. B. G. thanks...". Put sponsor acknowledgments in the unnumbered footnote on the first page.

References

Please number citations consecutively within brackets [15]. The sentence punctuation follows the bracket [16]. Refer simply to the reference number, as in [17]—do not use "Ref. [17]" or "reference [17]" except at the beginning of a sentence: "Reference [17] was the first ..."

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors' names; do not use "et al.". Papers that have not been published, even if they have been submitted for publication, should be cited as "unpublished" [18]. Papers that have been accepted for publication should be cited as "in press" [19]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [20].

References

[1] TODO.
[2] TODO.
[3] TODO.
[4] TODO.
[5] TODO.
[6] TODO.
[7] TODO.
[8] TODO.
[9] TODO.
[10] TODO.
[11] TODO.
[12] TODO.
[13] TODO.
[14] TODO.
[15] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.
[16] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.
[17] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.
[18] K. Elissa, "Title of paper if known," unpublished.
[19] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.
[20] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].
[21] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.