USING SOFTWARE PHYSICS TO SIZE WORKLOADS WITH SAS

Kenneth W. Kolence
Institute for Software Engineering
P.O. Box 637, Palo Alto, California 94302

ABSTRACT

This paper discusses the use of software work vectors as the fundamental form for workload characterization purposes. Software work is one of the basic properties of software physics, and has the property of being independent of processor speed, configuration design, and is unaffected by other workload elements. The general vector form of characterization is defined. Equations for calculating work are given, and examples of its uses considered. The relation of this form to the data used for synthetic benchmarks and workload analyses is briefly reviewed.

***

QUANTIFICATION REQUIREMENTS

Workload characterization is the quantification of the important characteristics of a workload or any arbitrary portion of it. The word "important" implies a use or purpose for the quantitative description, and so several different forms of characterization may be needed according to the intended use or uses. If these different characterizations are to be usable together, they must all spring from a single fundamental form of quantitative description. Further, this fundamental form must be completely applicable to all workload and machine architectures, and must be independent of the vagaries of data provided by current instrumentation. Software physics provides both the fundamental and derivative characterization forms which meet these and other important theoretical and practical criteria.

SOFTWARE PHYSICS BACKGROUND

Software physics is built up from the assumption that only three fundamental measurable properties exist: software work, execution time, and storage occupancy. These properties, when measured, are associated with some combination of computing equipment and some arbitrary collection of executable code and associated operands. The collection of computing equipment may be thought of either in terms of configurations and subconfigurations, or in terms of equivalent classes of devices; e.g., all disks, all tapes, all cpu's, etc. Arbitrary collections of executable code and associated operands are called "units of software." Examples include job steps, subroutines, single instructions, jobs, full applications, the TSO workload, the batch workload, the operating system, and the full workload.

A given measurement must reference the specific configuration/subconfiguration or equipment class, and the unit of software being measured. The value of the measurement in software physics will represent either one of the fundamental properties or a property derived in terms of these fundamental properties. For example, if the rate of data transfer is measured this is equivalent to measuring the software work \( W \) and the execution time \( T_x \), and obtaining a value for software power \( P = W + T_x \).

Software physics (or any other general theory concerned with the meaning and use of computer measurements) must have what is called a "precisely measurable notation" with which to represent the properties described. Such a notation has the characteristics that a value represented in the notation will be unambiguously defined. That is, two different people can be given the same symbol, and they will be able to independently make measurements and obtain the same numerical results. If a theory is not developed with a precisely measurable notation, then the theory is expressed ambiguously and will not generally lead to the same results in the hands of different people.

The full structure and symbolism of the software physics notation cannot be considered in this paper, although it is fully developed elsewhere. Further, this paper is primarily limited to considerations involving the software physics property "software work." The notational definitions required for these purposes will thus be introduced only within this context.

SOFTWARE WORK

The formal definition of software work, including the definition of its units, is as follows:

A processor performs one unit of software work on a storage device when it alters the symbol state of one byte of that storage device.

This definition has the operational difficulty that the instrumentation required to determine if the contents of a byte of storage has actually been changed is rather horrendous. As a result, the approximation used as an operational definition assumes that a "transfer of a byte" to storage always results in an alteration of the storage contents. One then has as an operational definition:

A processor performs one unit of software work on a storage device when it transfers one byte to that storage device.
SOFTWARE WORK CALCULATIONS

As an example, a tape drive writing a 1000 byte block onto tape does 1000 w or 1 KW on the magnetic tape. The same drive reading a 1000 byte block ultimately does 1000 w or 1 KW on main storage. Notice that the units in which software work are measured are called "work(s)", so that 1000 works are expressed as one kilowork (KW), a million as one megawork (MW), and a billion as one gigawork (GW).

Work can be measured directly with a hardware monitor, or calculated with various approximations. For I/O devices, the most generally useful approximation equation for use with SMF data is:

\[(I/O \text{ work}) = (\#\text{EXCP's})(\text{Av. Block Size})\]

Although the property "software power" has not yet been precisely defined, it is used in the most generally useful approximation equation for determining cpu work. Since the quantity "cpu seconds" is commonly available, under the typically valid assumption that cpu power is constant, one has:

\[(\text{cpu work}) = (\text{Av. cpu power})(\text{cpu seconds})\]

Typical values for the average cpu power might be 20 MW/s for a 168 and 7 MW/s for a 168 mad 1.

SOFTWARE WORK NOTATION

The definition of software work explicitly mentions a processor, and implicitly refers to some unit of software. A processor is any device or collection of devices which can execute an instruction, and any executable instruction or collection of such instructions along with their operands defines a unit of software. Thus, to notationally represent a quantity of software work, it is necessary to identify the unit of software and the processor.

For the purposes of this paper, the term "processor" will be considered as meaning a class of equivalent devices, such as disks, tapes, cpu's, printers, terminals, etc. A unit of software will be represented by the symbol S in general, and by the symbol L when the unit of software which is the full workload is being considered. The structure of the software physics notation in general is functional; i.e.,

Property (Software Unit, Processor)

The property software work is represented by the symbol \( W \). If one wishes to represent the software work done for the full workload \( L \) by the cpu, one writes

\[ W(L,\text{cpu}) \]

The software work done by the disk drives for \( L \) is written as

\[ W(L,\text{disks}) \]

The equivalent symbol for tape drives is

\[ W(L,\text{tapes}) \]

Printer work for the full workload is

\[ W(L,\text{ptr}) \]

All of the above symbols relate to the full workload \( L \). The reader should easily be able to construct the symbols for the full workload on other equipment classes. Similar symbols, with \( S \) replacing \( L \), define the software work done by units of software less than the full workload. For example, one might write \( W(S,\text{cpu}) \) for the cpu work done by some \( S \). If the unit of software being measured is identified by the name \( \text{SORT} \), one would write \( W(\text{SORT},\text{cpu}) \) for the cpu work. Likewise, \( W(\text{SORT},\text{disks}) \) and \( W(\text{SORT},\text{tapes}) \) would represent the disk and tape work done for \( \text{SORT} \).

ADDITIVE WORK PROPERTIES

Software work stands alone among the fundamental properties of software physics in that it is the only extensive property; i.e., the whole is the arithmetic sum of the parts, no matter how the parts are taken. Mathematically, examples of this extensiveness might be represented by the equations

\[ W(L,\text{cpu}) = W(S_1,\text{cpu}) + W(S_2,\text{cpu}) + \cdots \]

where the workload \( L \) is composed of \( S_1, S_2, \text{etc.} \)

If the symbol \( W(L,\_\,) \) is used to represent the work done by the entire configuration, one can write

\[ W(L,\_\,) = W(S_1,\_\,) + W(S_2,\_\,) + \cdots = W(S_1,\text{cpu}) + W(S_2,\text{cpu}) + 4W(S_1,\text{disk}) + W(S_2,\text{disk}) + \cdots + 4W(S_1,\text{tapes}) + W(S_2,\text{tapes}) + \cdots + \cdots \]

From these samples of the extensiveness characteristics of software work, it follows that software work also has the extremely important properties that:

1. For a given unit of software, software work by equipment class is unaffected by other software units in the workload, whether run concurrently or not.
2. By equipment class, the software work done by a unit of software is unaffected by the exact configuration into which equipment is connected.

Beyond this, the quantity of software work done for a unit of software within a given class of equipment is unaffected by the speed of the devices in that class. Clearly, replacing a tape drive with a faster tape drive does not affect the amount of work done, and similarly with disks. Within a family of compatible cpu's,
such as the 360/370 family, the speed of the cpu will not affect the cpu work done for a given unit of software.

WORK VECTOR CHARACTERIZATION

From all of these characteristics of software work, one draws the strong conclusion that software work characterizes a unit of software in a fashion independent of both the equipment characteristics and the workload environment. For this reason, the characterization of a workload or other arbitrary unit of software is most fundamentally given in terms of software work by equipment class.

In general, though not necessarily, a given unit of software can cause work to be done in each equipment class represented in the configuration. The most convenient and correct way to represent the total work is in the "Software work vector" form. Following normal mathematical notation convention, the vector form of a quantity is represented by use of an arrow over the symbol. A software work vector for S on the full configuration would thus be represented as

\[ \mathbf{W}(S) = \begin{bmatrix} W(S,cpu) \\ W(S,disk) \\ W(S,tapes) \end{bmatrix} \]

Since software work is an extensive property, the absolute value of the vector, \(|\mathbf{W}(S)\)| is simply the arithmetic sum of the components. That is:

\[ |\mathbf{W}(S)| = \sum W(S,i) \]

If one divides each component of a software work vector by the total work \(W(S)\), one obtains

\[ \mathbf{W}(S) = \frac{W(S,cpu)}{W(S)} \cdot \mathbf{W}(S,disk) \cdot \mathbf{W}(S,tapes) \]

Following normal scalar-vector multiplication rules, one can therefore also write the quantity \(\mathbf{W}(S)\) as:

\[ \mathbf{W}(S) = \mathbf{W}(S) \cdot \begin{bmatrix} \mathbf{W}(S,cpu) \\ \mathbf{W}(S,disk) \\ \mathbf{W}(S,tapes) \end{bmatrix} \]

The vector composed of the components \(\mathbf{W}(S,cpu), \mathbf{W}(S,disk), \mathbf{W}(S,tapes)\), etc., is called a unit vector, because the arithmetic sum of the components will always total to one from the method of construction.

The unit vector form of a software work vector, referring to the product of the total work and the unit vector, is the fundamental form of characterization of any unit of software, including the full workload.

At this point, it is necessary to re-emphasize the definition of the term "unit of software." It is both the executable code and the associated data. A program is normally thought of only in terms of the code plus internal constants and tables. A unit of software also includes a specific set of data. Thus, the same code with different data constitutes a different though similar unit of software. In other words, the unit vector form characterizes the combination of code and data, not just the code alone. A small amount of reflection should convince the reader that this must be the case for any form of workload characterization, since the data input to a program can radically alter its quantitative behavior.

However, nothing in the definition of a unit of software requires that the code and data referred to compose a single run of the unit of software. Thus, a given set of executable code and all of the runs over different data in, say, a given month equally well represents a unit of software. Thus, typical monthly reports on an application which give the cpu seconds and EXCP's by device class can be converted to the unit vector form by the approximations given earlier. So long as the code remains unchanged and the mix of data processed is relatively the same from month to month, the reports will be comparable.

If one forms the unit vector form of characterization for a typical application over a several month period, the total work may very considerably but usually the unit vector itself is constant or very nearly so. Such a software unit is called a linear software unit. Conversely, a non-linear software unit is one in which the unit vector changes with the total work performed. Thus, among the other valuable properties of this form of characterization, it permits one to describe the quantitative behavior of a given set of code under different data loads. Likewise, the complete unit vector form permits the characterization of a given set of code under special mixes of data; e.g., each different record type processed.

EXAMPLES OF WORK VECTOR USES

An example using real data is probably useful at this point. It represents a large invoice control application for a major U.S. corporation. It is run on a 370/158, and represents perhaps 10% of their batch workload. The average disk and tape block sizes for the application were determined using type 14 and 15 SMF records, and the cpu power value for their 158 was measured by a hardware monitor. Using data from their job-accounting package, and the software work approximations given earlier, the following work vectors were obtained over a three month period. Just the cpu, disk, and tape work components are given for simplicity. Also, due to round-off error, the unit vectors do not precisely equal 1.0000.
The problem of characterizing or sizing an "average job" is an important one to solve for purposes of throughput comparisons, either over time within an installation or between installations. Given the unit vector form, the computation of the average job size is trivial. The average job size may be computed for the full batch workload, for applications or sets of applications, for two, or other workloads for which data is available. To compute it, one simply divides the total work by the number of jobs processed.

For example, using the above data, assume 80 different jobs made up the application over the three month period. Then the average job size may be computed for each month, and also the average over the three month period. Taking the latter (the last column in the data above), the average job size \( W(S,\bar{\Psi}) \) is:

\[
\bar{W}(S,\bar{\Psi}) = \frac{108.36 \text{GW}}{80} = 1.355 \text{GW}
\]

It is often possible to carry the characterization to a further level of detail and value. In many instances, the total work to be done is purely determined by the quantity of input items processed and the unit vector is constant. In these instances, dividing the total work \( W(S,\bar{\Psi}) \) by the number of input items will give the "work per transaction" as a constant value. Thus, software work workload forecasts based on the number of transactions forecasted by the application user can easily be constructed.

Using the above example of application work vectors, the number of invoices processed each month, and the work per invoice is as given below:

<table>
<thead>
<tr>
<th>Month 1</th>
<th>Month 2</th>
<th>Month 3</th>
<th>Avg/Month</th>
</tr>
</thead>
<tbody>
<tr>
<td>Work</td>
<td>107.50GW</td>
<td>106.85GW</td>
<td>110.74GW</td>
</tr>
<tr>
<td>Invoices</td>
<td>19,228</td>
<td>18,405</td>
<td>18,517</td>
</tr>
<tr>
<td>Work/invoice</td>
<td>5.59 MW</td>
<td>5.60</td>
<td>5.98</td>
</tr>
</tbody>
</table>

In the data given here, a trend towards increasing work per transaction seems to be apparent, although the average still seems close enough to use for forecasting purposes. Clearly, in actual practice one would want to monitor the data to determine if this is an actual trend. If so, the causes should be determined and corrective action taken if possible because this corresponds to an increase in the cost per invoice processed. This illustrates another useful property of this characterization form: it provides an excellent basis for management control on the productivity (work per $) of an application.

In practice, changes in the work per transaction may come from several sources. Data mixes may actually be changing somewhat over time. runaway may be increasing (or decreasing) as a proportion of production. Maintenance changes may be altering the work per transaction. Additional reports may be being added. All of these activities are reflected properly in the unit vector form of characterization.

There are many other uses of this form of workload characterization in software physics, such as performance measurement and prediction, their use in giving performance specifications for new applications to be developed, equipment planning, and determining the capacity remaining in a configuration. Unfortunately, these all require a much more extensive treatment of the software physics property called software power. Full details of many of these uses are given in reference 1.

SOFTWARE PHYSICS AS A UNIFYING CONCEPT

One last comment is in order before closing, however. If software physics truly is a general theory, then one of the most fundamental characteristics which the theory must display is that of "unification." This is, any research which is based on quantitative data obtained by measurement, and which further is empirically found to be valid, must be capable of being described in terms of the fundamental properties of software physics. Doing so will automatically unify the effort with the other portions of software physics. If this unification cannot be done, then software physics theory is incorrect and must be at least adjusted to accommodate the new knowledge.

Several outstanding examples exist of the ability of software physics to unify hitherto disparate knowledge. In fact, reference 1 treats most of the good practices found in installations for workload forecasting, performance analysis, equipment planning and configuration design, charge-back systems, etc. The concern in software physics theory to demonstrate the unification of all these practices into a consistent body of knowledge is often confused with the fact that such practices are already known. This results in the criticism that "there is nothing new in software physics." However, the unification process itself is new, and the results of restating these practices in software physics terms has led to better quantification, formalization, and in many instances much simpler descriptions of these practices. And, of course, a much better understanding of why in fact they do work.

Recognizing then that the ability to unify existing but disparate knowledge into software physics is probably the crucial test on the
validity of the theory, the concepts of synthetic benchmarks and workload clustering analysis need to be discussed. Considerable excellent work has been done by many people in this field, particularly at MITRE and the University of Maryland. All of it is based on measures such as cpu seconds and I/O's (EXCP's). Since these are directly related to software work by means of the approximation equations given earlier, all of this work can easily be translated directly into software physics terms. When this is done, one finds that all of the effort is based on the concepts of a software work vector as presented herein. Because of this unification with software physics, such work can now be directly related to all of the other aspects of software physics and is indeed on a firm theoretical foundation.

References