Efficient Scalable Thread-Safety-Violation Detection

CSE 513

Melek Nurten YAVUZ

Advanced Topics in Operating Systems

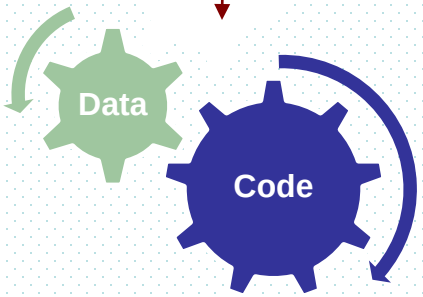
Aralık 2019

İçerik

- Çalışmanın Amaç ve Nedenleri
- Çalışmanın Tasarımı
- Çalışmanın Gerçeklenmesi
- Diğer Çalışmalarla Karşılaştırma
- Sonuç

Çalışmanın Amaç ve Nedenleri

- Eşzamanlılık hatalarını bulmak ve hata ayıklamak zordur.
- Hatalar sık sık şirket içi zorlu testlerden kaçarlar ancak sonuçta üretimde büyük çaplı kesintiler meydana getirirler.
- Bu tip projelerde asıl sorular şunlardır:



1. Binlerce mühendislik ekibi tarafından geliştirilen kodun nasıl kullanılacağı
2. Senkronizasyon mekanizmalarının çeşitliliği, false pozitiflerin nasıl az / hiç olmadığı, nasıl test edileceği
3. Aşırı test kaynak tüketiminin nasıl önleneceği

- Dictionary sınıfının uygulanması, birden fazla iş parçacığının aynı anda ContainsKey gibi okuma işlemlerini çağırmasına izin verir, ancak Add gibi yazma işlemlerinin yalnızca özel bağlamlarda çağrılmasını gerektirir.

```
1  // Dictionary dict
2  // Thread 1:
3  dict.Add(key1, value)
4  // Thread 2:
5  dict.ContainsKey(key2)
```

Figure 1. A thread-safety violation (TSV) bug

Çalışma Tanımı

Bu makale TSVD'yi sunmaktadır:

Çalışma, karmaşık multi-thread yapıları kullanılmış olan ve asenkron programlama modelleri olan .net programları için bug tespiti sağlar.

Bunu da, tasarım uzayında gezinmek ve bug tespiti için bir başlangıç noktası sağlayarak yapıyor.

Tespit için yazılımın build aşamasında test taraflarına entegre olarak trap methodu çalıştırır.

TSVD Algoritma Tasarımı

```
1  OnCall(thread_id, obj_id, op_id){
2    check_for_trap(thread_id, obj_id, op_id)
3    if(should_delay(op_id)){
4       set_trap(thread_id, obj_id, op_id)
5       delay()
6       clear_trap(thread_id, obj_id, op_id) }}
```

Figure 5. The trap mechanism used by TSVD and its variants

Trap mekanizması şu şekilde çalışır.

- OnCall yöntemini çağıran bir method düşünün. 3. satırda should_delay işlevi tarafından seçilen bazı aramalarda, iş parçacığı geçerli method çağrısını satır 4'teki bazı genel tabloya kaydederek yakalayın.
- Tuzak, iş parçacığının, nesnenin ve işlemin tanımlayıcılarının üçlüsü tarafından tanımlanır. Ardından, iş parçacığı, tuzak "ayarlanmış" olduğu süre boyunca uyku için gecikme yöntemini çağırır.

TSVD Algoritma Tasarımı

Tasarımsal Detaylar

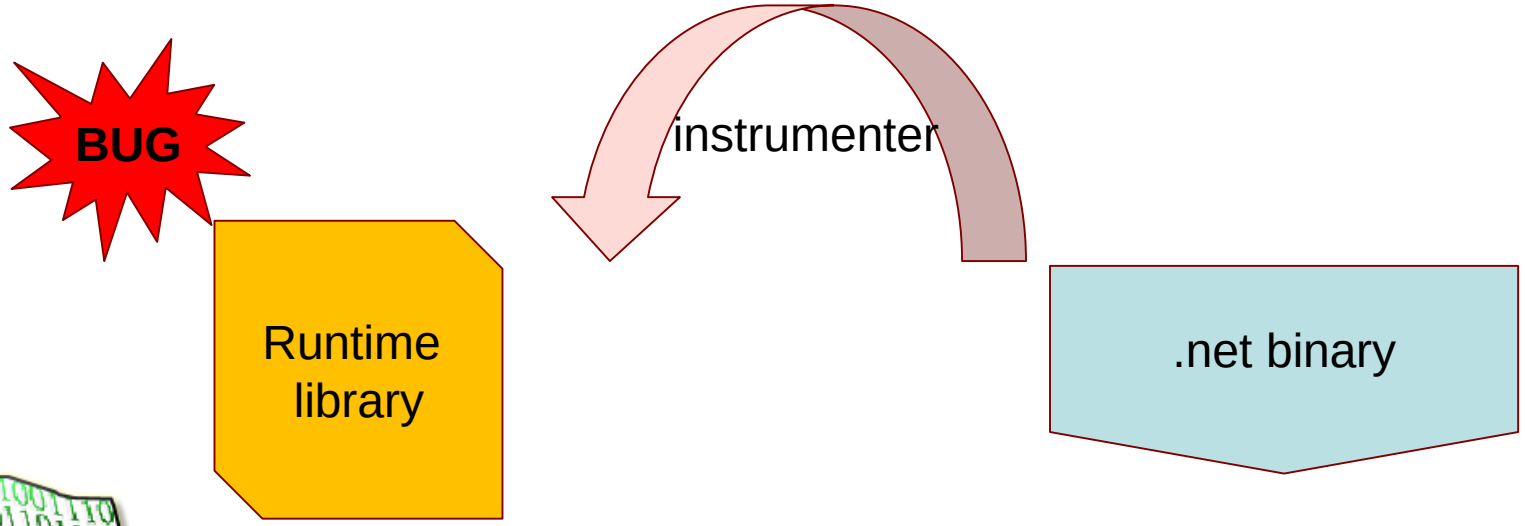
TSVD ve değişkenleri should_delay ile ilgili iki ana tasarım sorusuna verdikleri cevaplarda farklılıklar göstermektedir:
1. Gecikmeler nereye enjekte edilir? Hangi program yerleri should_delay için gecikmeli enjeksiyon önerebilir.

TSVD çakışmaların olabileceği Ploc terimini kullandığı bir hesaplamayla bir çift tehlikeli location seçer. Ve bugların burada olmasını umar. (Synchronization operations like forks, joins, barriers, and locks could lead to sequential execution phases during the execution of a concurrent program)
2. Ne zaman gecikmeler enjekte edilmeli?

TSVD delay planını ve enjeksiyonu aynı zamanda aynı anda yapıyor. Bu da test kaynaklarını en iyi şekilde yönettiğini kanıtlıyor.

Tasarımsal Detaylar

TSVD'nin iki ana bileşeni bulunur.
Runtime kütüphanesi ana algoritma olarak implement edilmiştir.
TSVD instrumenter olan diğer bileşen ise verilen .net binarysini runtime kütüphaneye otomatik olarak birleştirir.



Diğer Çalışmalarla Karşılaştırma

Sonuclara göre en fazla bug tespitini TSVD yapmıştır. 42 bug buldu ve ikinci runda 11 ek bug buldu. DataCollider ve Dynamic Random ın TSVD yi capasite olarak yakalayabilmesi için birden fazla çalışması gereklidir.Bu da cost u artırmaktadır. Overhead=time

	# bug				
	Total	Run1	Run2	overhead	# delay
DataCollider	25	22	3	378%	77402
DynamicRandom	13	6	7	178%	31456
$TSVD_{HB}$	41	25	16	310%	3328
TSVD	53	42	11	33%	22632

Table 2. Comparing TSVD with other detection techniques.

Diğer Çalışmalarla Karşılaştırma

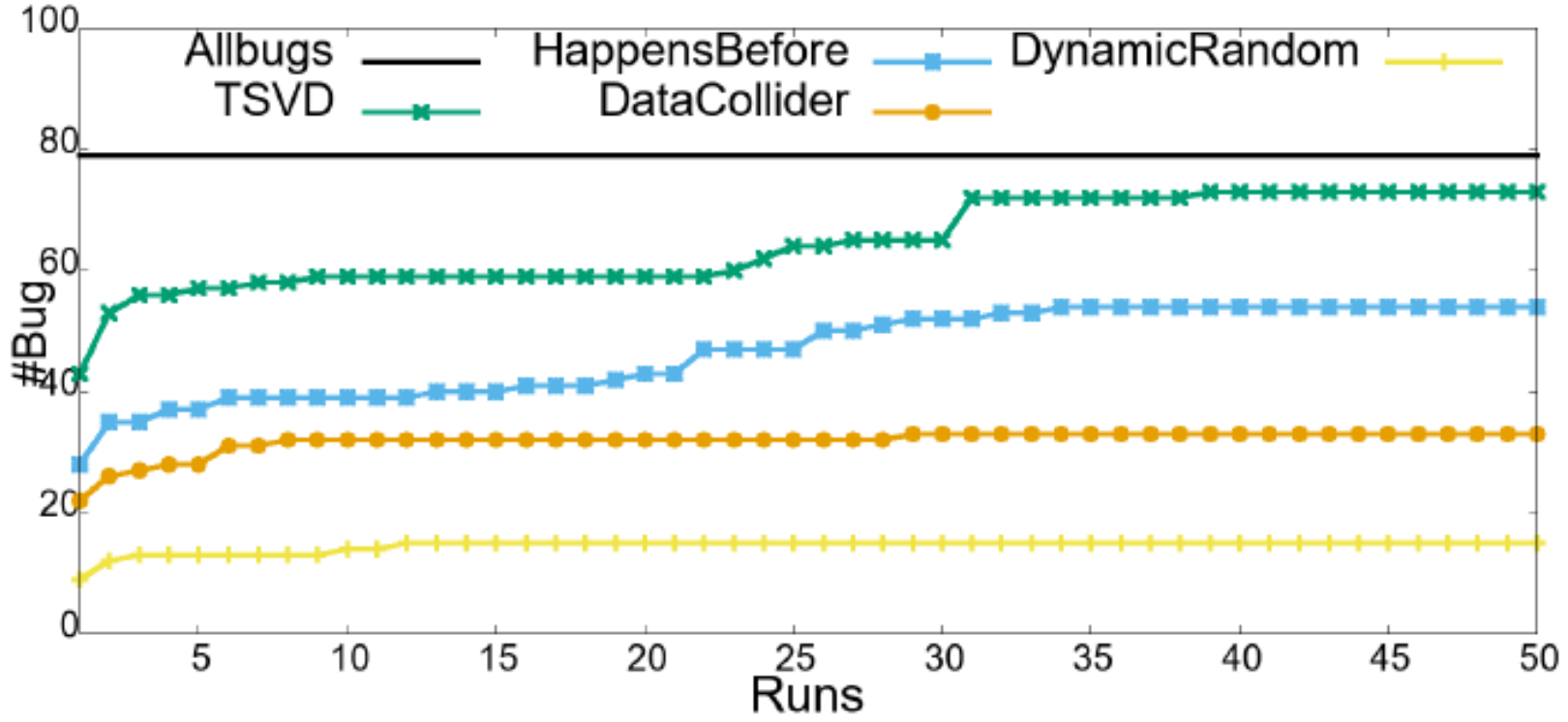


Figure 8. Number of bugs found after more runs

Sonuç

Test Targets	
# of projects	1,657
# of test modules	$\approx 43K$
# of binaries	$\approx 89K$
# of tests	$\approx 122K$
Bugs found	
# of unique bugs (location pairs)	1,134
# of unique bug locations	1,180
# of unique stack trace pairs	21,013
% of projects with bugs	9.8%
% of modules with bugs	1.9%
Bugs properties	
% of read-write bugs	48%
% of same location bugs	34%
% of bugs in `async` code	70%
Avg. (Median) occurrence of a bug location	36(4)
Avg. (Median) # stack trace pairs/bug	18.5(3)
Avg. Stack depth	9.1
% of `Dictionary` bugs	55%
% of `List` bugs	37%

Table 1. Summary of bugs found by TSVD tools.

Sorular

1) Thread senkronizasyon sağlama operasyonlarından ikisini yazınız.

Cevap: locks, joins

2)Bir programda senkronizasyon bazlı buglar hangi işlemlerde ortaya çıkabilir, örnek veriniz.

Cevap: Okuma / Yazma işlemlerinde


Teşekkürler!