Probabilistic Simulation with Computation Costs

Louis-Claude CANON

March 27, 2013

The objective is to simulate the behavior of static and dynamic approaches with probabilistic communication and computation costs (either \(d\) with a probability \(p_d\) or \(D\) for the communication and \(c\) with a probability \(p_c\) or \(C\) for the computation).

1 Tree construction

```r
# Function to build a tree

tree_build <- function(n, d, c) {
    parent <- array(0, n)
    soonest <- array(0, n)
    soonest[n] <- c
    for (i in (n - 1):1) {
        minCost <- Inf
        for (j in (i + 1):n) {
            currentCost <- soonest[j] + d
            if (currentCost < minCost) {
                minCost <- currentCost
                bestSucc <- j
            }
        }
        soonest[i] <- minCost + c
        parent[i] <- bestSucc
    }
    return(parent)
}
```

2 MC simulator for static approaches

Now, let’s define a function for evaluating trees:

```r
# Function to simulate static approaches

static_MC <- function(T_orig, rdistd, rdistc, iterations) {
    result <- NULL
    for (k in 1:iterations) {
        T <- T_orig
        Ld <- array(0, length(T))
        Lc <- array(0, length(T))
        while (any(T != 1:length(T))) {
            # Detail of the simulation process
        }
    }
}
```
# Select leaves
leaves <- which(!(1:length(T) %in% T))
# Find the next that is ready
current <- leaves[which.min(Lc[leaves])]
if (T[current] != 0) {
    Ld[T[current]] <- max(Ld[T[current]], Lc[current]) + rdistd(1)
    Lc[T[current]] <- max(Ld[T[current]], Lc[T[current]]) + rdistc(1)
} else {
    T[current] <- current
    result <- c(result, max(Lc))
} else {
    return(result)
}

n <- 8
iterations <- 1e+05
rdistd <- function(n, p = 0.5, d = 11, D = 29) {
    return(sample(c(d, D), n, replace = TRUE, prob = c(p, 1 - p)))
}
rdistc <- function(n, p = 0.5, d = 0, D = 0) {
    return(sample(c(d, D), n, replace = TRUE, prob = c(p, 1 - p)))
}
table(static_MC(tree_build(n, 1, 0), rdistd, rdistc, iterations))/iterations

##
## | 33 | 44 | 51 | 62 | 69 | 80 | 87 |
## |----|----|----|----|----|----|----|
## | 0.00752 | 0.00808 | 0.18582 | 0.01637 | 0.47867 | 0.00796 | 0.29558 |

3 MC simulator for dynamic approaches

dyn_MC <- function(n, rdistd, rdistc, iterations) {
    result <- NULL
    for (k in 1:iterations) {
        comm <- rdistd(n/2) + rdistc(n/2)
        available <- FALSE
        while (length(comm) > 1 || available) {
            current <- which.min(comm)
            if (available)
                comm[current] <- comm[current] + rdistd(1) + rdistc(1) else comm <- comm[-current]
            available <- !available
        }
        result <- c(result, comm[1])
    }
    return(result)
}
dyn_com_MC <- function(n, rdistd, rdistc, iterations) {
    result <- NULL

2
```r
for (k in 1:iterations) {
    comm <- rdistd(n/2) + rdistc(n/2)
    while (length(comm) > 1) {
        current <- which.min(comm)
        first <- TRUE
        if (current < length(comm) && comm[current + 1] == Inf && current >
            1 && comm[current - 1] == Inf && rbinom(1, 1, 0.5) == 1)
            first <- FALSE
        if (current < length(comm) && comm[current + 1] == Inf && first) {
            comm[current] <- comm[current] + rdistd(1) + rdistc(1)
            comm <- comm[-(current + 1)]
        } else if (current > 1 && comm[current - 1] == Inf) {
            comm[current] <- comm[current] + rdistd(1) + rdistc(1)
            comm <- comm[-(current - 1)]
        } else {
            comm[current] <- Inf
        }
        result <- c(result, comm[1])
    }
    return(result)
}

Some preliminary tests (binomial and dynamic are equivalent with no variation):

```n <- 8
iterations <- 100
rdistd <- function(n, p = 0.5, d = 11, D = 11) {
    return(sample(c(d, D), n, replace = TRUE, prob = c(p, 1 - p)))
}
rdistc <- function(n, p = 0.5, d = 13, D = 13) {
    return(sample(c(d, D), n, replace = TRUE, prob = c(p, 1 - p)))
}
table(static_MC(tree_build(n, 1, 0), rdistd, rdistc, iterations))/iterations
##
## 72
## 1
table(dyn_MC(n, rdistd, rdistc, iterations))/iterations
##
## 72
## 1

Second test (checking that both dynamic methods are equivalent with two values):

```n <- 2
iterations <- 1e+05
rdistd <- function(n, p = 0.5, d = 11, D = 29) {
    return(sample(c(d, D), n, replace = TRUE, prob = c(p, 1 - p)))
}
rdistc <- function(n, p = 0.5, d = 13, D = 31) {
  return(sample(c(d, D), n, replace = TRUE, prob = c(p, 1 - p)))
}
table(dyn_MC(n, rdistd, rdistc, iterations))/iterations
##
## 24 42 60
## 0.2488 0.5003 0.2510
table(dyn_com_MC(n, rdistd, rdistc, iterations))/iterations
##
## 24 42 60
## 0.2498 0.5005 0.2496

4 Comparisons

compare_approaches <- function(n, iterations, rdistd, rdistc) {
  bino <- static_MC(tree_build(n, 1, 0), rdistd, rdistc, iterations)
  fibo <- static_MC(tree_build(n, 1, 1), rdistd, rdistc, iterations)
  dyn <- dyn_MC(n, rdistd, rdistc, iterations)
  dyn_com <- dyn_com_MC(n, rdistd, rdistc, iterations)
  plot.ecdf(bino, verticals = TRUE, cex = 0.3, xlim = range(c(bino, fibo, dyn, dyn_com)))
  plot.ecdf(fibo, verticals = TRUE, cex = 0.3, add = TRUE, col = 2)
  plot.ecdf(dyn, verticals = TRUE, cex = 0.3, add = TRUE, col = 3)
  plot.ecdf(dyn_com, verticals = TRUE, cex = 0.3, add = TRUE, col = 4)
  legend("bottomright", c("bino", "fibo", "dyn", "dyn_com"), col = 1:4, pch = 1)
}

n <- 64
iterations <- 10000
rdistd <- function(n, p = 0.5, d = 11, D = 29) {
  return(sample(c(d, D), n, replace = TRUE, prob = c(p, 1 - p)))
}
rdistc <- function(n, p = 0.5, d = 13, D = 31) {
  return(sample(c(d, D), n, replace = TRUE, prob = c(p, 1 - p)))
}
compare_approaches(n, iterations, rdistd, rdistc)
With computation, Fibo is better than bino (as expected) and even dynamic approaches.
Now, if the variability is higher, both dynamic approaches should be better:

```r
n <- 64
iterations <- 10000
rdistd <- function(n, p = 0.5, d = 11, D = 89) {
  return(sample(c(d, D), n, replace = TRUE, prob = c(p, 1 - p)))
}
rdistc <- function(n, p = 0.5, d = 13, D = 97) {
  return(sample(c(d, D), n, replace = TRUE, prob = c(p, 1 - p)))
}
compare_approaches(n, iterations, rdistd, rdistc)
```
Even with high variability, Fibo manage to be better than dyn. With a even larger variability:

\[ \text{n <- 64} \]
\[ \text{iterations <- 10000} \]
\[ \text{rdistd <- function(n, p = 0.5, d = 11, D = 211) {} \]
\[ \quad \text{return(sample(c(d, D), n, replace = TRUE, prob = c(p, 1 - p)))} \]
\[ \text{}} \]
\[ \text{rdistc <- function(n, p = 0.5, d = 13, D = 223) {} \]
\[ \quad \text{return(sample(c(d, D), n, replace = TRUE, prob = c(p, 1 - p)))} \]
\[ \text{}} \]
\[ \text{compare_approaches(n, iterations, rdistd, rdistc)} \]
Fibo and dynamic are equivalent. Fibo is still worse than dynamic with non-commutative operations.

5 MC simulator for static approaches respecting the initial order

```r
# This function assumes that the order in the tree is the same as the
# communication order
static_order_MC <- function(T, rdistd, rdistc, iterations) {
    result <- NULL
    for (k in 1:iterations) {
        Ld <- array(0, length(T))
        Lc <- array(0, length(T))
        # Code
    }
    result
}
```
for (i in 1:(length(T) - 1)) {
    Ld[T[i]] <- max(Ld[T[i]], Lc[i]) + rdistd(1)
    Lc[T[i]] <- max(Ld[T[i]], Lc[T[i]]) + rdistc(1)
}  
result <- c(result, max(Lc))
return(result)
}

n <- 8
iterations <- 1e+05
rdistd <- function(n, p = 0.5, d = 11, D = 29) {
    return(sample(c(d, D), n, replace = TRUE, prob = c(p, 1 - p)))
}
rdistc <- function(n, p = 0.5, d = 0, D = 0) {
    return(sample(c(d, D), n, replace = TRUE, prob = c(p, 1 - p)))
}
table(static_MC(tree_build(n, 1, 0), rdistd, rdistc, iterations))/iterations
##
## 33 44 51 62 69 80 87
## 0.00786 0.00781 0.18710 0.01640 0.47702 0.00795 0.29586

table(static_order_MC(tree_build(n, 1, 0), rdistd, rdistc, iterations))/iterations
##
## 33 51 69 87
## 0.00727 0.19346 0.49356 0.30571

On a larger scale:

n <- 64
iterations <- 10000
rdistd <- function(n, p = 0.5, d = 11, D = 29) {
    return(sample(c(d, D), n, replace = TRUE, prob = c(p, 1 - p)))
}
rdistc <- function(n, p = 0.5, d = 13, D = 31) {
    return(sample(c(d, D), n, replace = TRUE, prob = c(p, 1 - p)))
}
bino <- static_MC(tree_build(n, 1, 0), rdistd, rdistc, iterations)
bino_order <- static_order_MC(tree_build(n, 1, 0), rdistd, rdistc, iterations)
fibo <- static_MC(tree_build(n, 1, 1), rdistd, rdistc, iterations)
fibo_order <- static_order_MC(tree_build(n, 1, 1), rdistd, rdistc, iterations)
plot.ecdf(bino, verticals = TRUE, cex = 0.3, xlim = range(c(bino, bino_order, fibo, fibo_order)))
plot.ecdf(bino_order, verticals = TRUE, cex = 0.3, add = TRUE, col = 2)
plot.ecdf(fibo, verticals = TRUE, cex = 0.3, add = TRUE, col = 3)
plot.ecdf(fibo_order, verticals = TRUE, cex = 0.3, add = TRUE, col = 4)
legend("bottomright", c("bino", "bino_order", "fibo", "fibo_order"), col = 1:4, pch = 1)
Respecting the order seems more sensible for the Fibonacci tree.

6 Conclusion

The code is ready for any distribution of computation and communication costs:

- Bino is good with low variability and low computation costs.
- Fibo is good with non-negligible computation costs.
- Dyn is good with non-negligible variability (large variability if computation costs are non-negligible).
- Dyn-com is good (although worse than dyn) with large variability and low computation costs.
Preparation for more consequent simulations:

- Is it possible to show that increasing or decreasing $p$ has the same effect as lowering the variability by decreasing the ratio $\frac{D}{d}$ with $p = 0.5$?

- Plot average of length (and standard deviation with error bar) for each method for a given $n$, $p = 0.5$, $d = 1$ and increasing values of $D$ (with a log-scale for the x-axis with $x = \frac{D}{d} - 1$). Draw a line between each average.

- How to study the variations of $\frac{c}{d}$ (from 0 to 1) and $\frac{C}{c}$ (3d plot)? In particular, is the case $\frac{D}{d} \neq \frac{C}{c}$ relevant?